# Chapter 5

# Cognitive Modelling in HCI Research
# Anna L. Cox and David Peebles

## 5.1 Overview

Over the last thirty years or so computers have evolved rapidly into powerful and complex systems that underlie virtually every aspect of modern life and, if current trends continue, it is likely that they will be even more pervasive in the future. With the increased embeddedness of computer technology into our society, the characteristics of users have diversified rapidly from a situation where the average user was often a white, middle-aged male with a particular educational and socio-economic background to one in which users of all ages, sexes, social and ethnic backgrounds, levels of education and computer knowledge are interacting with complex interfaces to computer systems. The range and complexity of people's interactions with computers have also grown rapidly over recent years, so that we now do many things via a computer (e.g., managing a bank account, paying household bills, shopping, organising a holiday) that would have been done in the high street just a few years ago.

These rapid developments present significant challenges to interface designers. As the range and sophistication of computer-based tasks have increased so have the interfaces that people are required to use and so the issue of how people perceive and process complex displays of information when carrying out tasks becomes ever more important. In order to understand these processes more closely, analysts have utilised the theories and methods of cognitive psychology—the study of human perception and information processing—to construct *cognitive models*: specifications of the mental representations, operations and problem solving strategies that occur during the execution of computer based tasks. These models can take several forms, from relatively general descriptions of the steps required to complete a task, to sophisticated computer simulations of users performing a task with an interface. Whatever the form, the process of cognitive modelling can benefit interface design by enabling analysts to develop a more precise and detailed understanding of human-computer interactions and in some cases to predict how users will behave. As such, much of the benefit of cognitive models is that they can be used early on in the design process as well as in the evaluation of existing designs. Cognitive models can also allow designers to identify and explain the nature of problems that users encounter and provide information concerning the cognitive and perceptual constraints on human performance to aid the design of systems that do not place too high processing demands on users or that can take advantage of particular aspects of users' skills and abilities.

In this chapter we describe two of the most commonly used cognitive modelling techniques and illustrate both with examples of their use to show the insights into human-computer interaction that each provides. The first approach consists of a family of analysis techniques based on a model of human infor-

mation processing and a related task analysis method proposed by Card, Moran and Newell (1983). The second, more recent approach employs *embodied cognitive architectures*—theories of cognition, perception and motor control implemented as software systems—to understand human-computer interaction by simulating it. Finally we point the reader towards a number of recent attempts to develop tools to integrate both of these approaches with the aim of facilitating the modelling process and making cognitive modelling techniques more accessible to a wider range of HCI practitioners.

## 5.2 Engineering Models

### 5.2.1 GOMS

The oldest and still arguably most widely used approach to modelling human-computer interaction is based on a model of human information processing and a task analysis method proposed by Card et al (1983). The models produced using this method are often called *engineering*, *predictive* or *zero-parameter* models because they are used to predict aspects of human performance with an interface or device *before* users are actually introduced to it. One benefit of such models is that they allow designers to evaluate different interface designs in terms of the speed and number of operations required to perform different tasks without actually having to build a complete system and get people to test them. This process differs from the cognitive modelling traditionally conducted in psychology which typically develops models with free parameters that can be adjusted to optimise the fit between the models' output and human data previously collected in experiments.

A core feature of this approach is the GOMS task analysis method. GOMS is an acronym formed from the four elements of the analysis: *Goals* (the aims of the user when interacting with the computer), *Operators* (the possible interactions that the interface allows, for example clicking and dragging with a mouse cursor, opening a text editor window, pressing a key on the keyboard etc.), *Methods* (sequences of subgoals and operators that can be used to achieve a particular goal) and *Selection rules* (the rules by which a user chooses a particular method from a number of alternatives for achieving a goal). The primary method for producing a GOMS task analysis is to break down tasks into a hierarchy of goals, *unit tasks* (basic learned sequences of integrated actions) and subtasks using the four elements outlined above. To illustrate the approach, below is a GOMS analysis, (adapted from Kieras, 1994) of the common task of deleting an object (e.g., a file or directory) using a graphical user interface such as Microsoft Windows or the K desktop environment (KDE) on Linux. All of the most commonly used graphical user interfaces use the "trash-can" metaphor for deleting objects according to which objects must be placed into a specific location in order to be deleted. All of the popular desktop environments also generally provide multiple ways for this goal to be achieved.

Method for goal: **delete object**
Step 1. Accomplish goal: move object to trash
Step 2. Return with goal accomplished

Method for goal: **move object [destination]**
Step 1. Accomplish goal: drag object [destination]
Step 2. Return with goal accomplished

Method for goal: **move object [destination]**
Step 1. Accomplish goal: send object [destination]
Step 2. Return with goal accomplished

Method for goal: **drag object [destination]**
Step 1. Locate icon for object on screen
Step 2. Move cursor to object icon location
Step 3. Hold left mouse button down
Step 4. Locate destination icon on screen
Step 5. Move cursor to destination icon
Step 6. Verify that destination icon indicates activation
Step 7. Release mouse button
Step 8. Return with goal accomplished

Method for goal: **send object [trash]**
Step 1. Locate icon for object on screen
Step 2. Move cursor to object icon location
Step 3. Hold right mouse button down
Step 4. Locate "Move to Trash" item on pop-up menu
Step 5. Move mouse cursor to "Move to Trash" item on pop-up menu
Step 6. Release mouse button
Step 7. Return with goal accomplished

The five methods above specify two ways to place objects in the trash: by dragging them there with the mouse or by sending the object there using a pop-up menu. There are alternative methods for doing this, for example by selecting the object and pressing the "Delete" key (KDE) or "Ctrl" and "D" keys together (Windows) but these are not included in this example. The first method defined above states that in order to delete an object the user must move the object to the "Trash" (KDE) or "Recycle Bin" (Windows). The next two methods then define two ways in which an object can be moved to destinations, the first by dragging it there with the mouse, the second by sending the object using a pop-up menu. The last two methods then describe the step-by-step actions (or "operators") that the user must perform in order to drag an object to a destination or send an object to the trash. Users may decide to use different ways to delete objects depending on specific internal or external factors, and these factors can be defined as selection rules, which typically are represented as conditional statements. For example, users may choose to drag an object to the trash if the trash-can is clearly visible but send it to the trash is the desktop is cluttered with windows and the trash-can is obscured from view. This situation may be represented by the following two rules:

IF the goal is to delete an object
AND the trash-can is visible
THEN use the *drag object* method

IF the goal is to delete an object
AND the trash-can is not visible
THEN use the *send object* method

Over the years GOMS has evolved into several variants that provide different analyses. Two widely used variants, KLM and CPM-GOMS are discussed below.

### 5.2.2 KLM

The Keystroke-Level Model (KLM) is a restricted version of GOMS that does not include goals or selection rules but simply specifies the sequence of operators and methods required to perform a task. The main

function of a KLM analysis is to predict the execution time of interactive tasks. All of the operators have a specific execution time and task completion time is calculated by summing the times spent executing the different operators. The eight standard operators with their associated estimated execution times (Kieras, 2001) are listed below:

**K** – Press a key on the keyboard. (0.28 sec).

**T(n)** – Type a sequence of n characters on the keyboard. (n $\times$ K sec).

**P** – Point the mouse to a target on the display. (1.1 sec).

**B** – Press or release the mouse button. (0.1 sec).

**BB** – Click mouse button. (0.2 sec).

**H** – Move hands between mouse and keyboard (or vice-versa). (0.4 sec).

**M** – Mental act of routine thinking or perception. (1.2 sec).

**W(t)** – Wait time for system response. (t msec).

For many of these operators estimated times depend on specific attributes of the user or environment. For example the 0.28 sec estimate for entering a keystroke is for a "typical" user and other estimates exist for expert typists (0.12 sec) and novices (1.2 sec). A KLM analysis of the two methods for deleting a file in the previous example (adapted from Kieras, 2001) is shown below.

Operator sequence: **drag object [trash]**

1. Point to file icon **(P)**
2. Press and hold mouse button **(B)**
3. Drag file icon to "Trash can" icon **(P)**
4. Release mouse button **(B)**
5. Point to original window **(P)**

Operator sequence: **send object [trash]**

1. Point to file icon **(P)**
2. Press and hold mouse button **(B)**
3. Point to "Move to Trash"item on pop-up menu **(P)**
4. Release mouse button **(B)**
5. Point to original window **(P)**

According to these analyses, both methods require three **P** and two **B** operators and so should both take $3\mathbf{P} + 2\mathbf{B} = 3 \times 1.1 + 2 \times 0.1 = 3.5$ sec to complete.

In the statistics and experimental design chapters of this book we have used an example of a study that compares a number of different interaction methods for composing a text message on a mobile phone. In this study, a KLM model is used to create a set of predictions of the time it would take a user to perform a task using each of the different interaction methods. These predictions (shown in table 5 of the statistics chapter) are used to form a hypothesis for the experimental study. The model enabled Cox, Cairns, Walton & Lee (forthcoming) to have confidence in their experimental results as they were able to explain both why the empirical data followed the pattern of their predictions, and where the KLM did not accurately reflect the behaviour of the human participants.

### 5.2.3 CPM-GOMS

The CPM in CPM-GOMS (John, 1988, 1996) can stand for either "cognitive-perceptual-motor" or "critical-path-method" and both alternative interpretations reveal a core assumption of the approach. The first indicates the theoretical basis of CPM-GOMS — the *model human processor* (MHP; Card et al, 1983), a simplified model of the three core components of human information processing: cognition, perception and motor control, together with a set of memories that store task knowledge, information from the environment and the current contents of cognitive processing. The MHP is governed according to a set of *operating principles* which determine how the processors behave and define such things as the time to make decisions, move a mouse cursor to a target or how performance time reduces with practice. MHP assumes that cognitive, perceptual and motor operators are performed in parallel and CPM-GOMS represents these parallel processes in the form of a *schedule chart* (sometimes known as a "PERT" chart). The second definition of CPM identifies a key feature of the analysis method — that of a *critical path* through the sequential dependencies between the three parallel processing streams that determines task execution time. A critical path is the sequence of processes with the longest overall duration, which consequently determines the shortest time possible in which the task can be completed. Figure 5.1 shows a CPM-GOMS model of the task of moving a mouse cursor to a button and clicking on it. The middle row depicts the cognitive operators with the perceptual and motor operators represented immediately above and below respectively. The number above each operator indicates the estimated execution time (in milliseconds) for that operator, with that of cognitive operators defaulting to 50 ms. According to the MHP all operators within a processing module are performed sequentially and lines between operators indicate sequential dependence between operators. The critical path through the chart is indicated by the bold lines and represents the dependencies between the cognitive, perceptual and motor operators. It is the accumulated times of these operators on the critical path that constitute the predicted total execution time for the task. In the example shown in Figure 5.1 this is $545 + 0 + 100 + 50 + 50 + 100 = 845$ ms. CPM-GOMS has been widely used to analyse a range of interactive behaviour, most famously when Gray, John and Atwood (1993) used CPM-GOMS in *Project Ernestine* to evaluate a new computer system for telephone operators at the NYNEX telephone company. The critical path produced by the CPM-GOMS analysis showed that the typical interaction when processing a call would actually take longer than with the existing system. Because of their predictions (which were subsequently validated by a field study), the new system was redesigned, thereby saving millions of dollars.

### 5.2.4 Limitations of the GOMS approach

Although the GOMS approach has been extremely useful in providing a range of techniques for analysing interactive behaviour and predicting execution times for a variety of tasks, it has a number of constraints (John & Kieras, 1996a). For example, analysts must assume that users are well practised, make no errors during the task or perform worse over repeated performance of a task due to fatigue. In addition, GOMS analyses require that reliable estimated times are available for all components in the task, which may not be the case for novel tasks or artefacts.

Cox et al's (forthcoming) KLM model referred to above provides us with an example of both of these limitations. First, the existing literature did not provide all the necessary information required to build the model. As no one had used KLM previously to model these novel interaction methods it was necessary to first conduct some small experiments to measure the average length of time it took to perform certain actions using the different interaction methods. Once gathered, these estimates could be used within the model. Second, as we have explained above, a KLM is a model of expert behaviour and as such does not model slips or mistakes. Of course, the performance of the participants in the experiment was not error free and as such was never going to match the predicted times exactly.

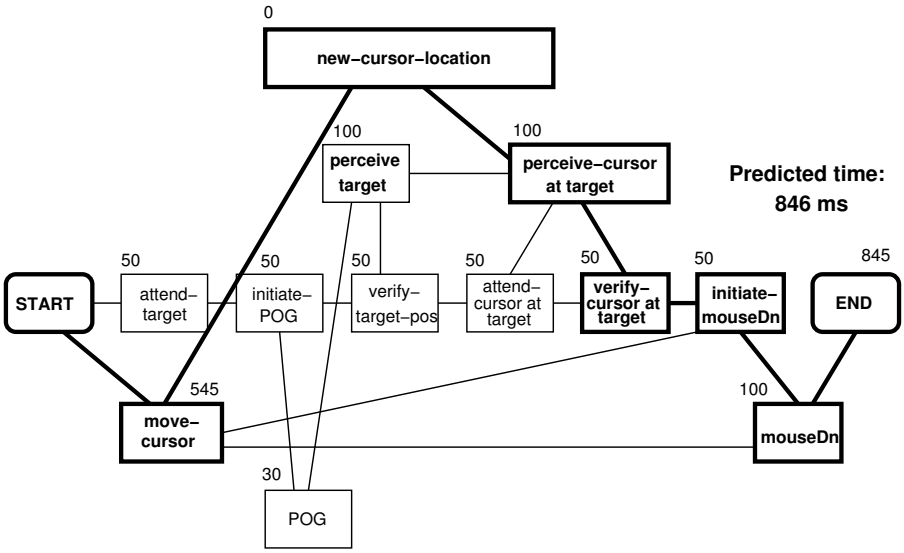


Figure 5.1: A CPM-GOMS model of a micro-strategy to move the mouse cursor to a button and clicking on it. From Gray & Boehm-Davis (2000)

## 5.3 Cognitive architectures

In contrast to a cognitive model, a cognitive architecture aims to provide an explanation of all aspects of cognition. One way of studying a computer based task in fine detail is to produce a computational model of the information and processes required to carry out the task that is built within a cognitive architecture. The architecture restricts the models that can be built within it thus ensuring that the model adheres to the psychological theories encapsulated by the architecture. Simulating interactions in this way is useful as it allows us to specify precisely the various perceptual, cognitive and motor processes involved in the interaction and to predict the effect of the human cognitive system (e.g., working memory) on behaviour. It is also useful to system designers as it has the potential to provide artificial users to allow us to test designs and judge whether one design is better in some way than another. This is possible because the models are themselves computer programmes which can be run. The resulting behaviour can be compared to human behaviour.

In HCI, we characterise humans as information processors (see Card, Moran & Newell, 1983) where information undergoes a series of ordered processes aimed at completion of a particular goal. This allows psychologists and cognitive scientists to view interaction with the computer as an information processing activity. Analysis of human-computer interaction in this way can support the work of interaction designers by helping them to understand what information requirements and resources are needed by human users when performing goal-based tasks.

Computational cognitive models are playing an increasingly important part in HCI research. Unlike the GOMS family of models which can model only expert performance, computational cognitive models allow researchers to show how users learn to use systems and also how users may perform in certain circumstances. Unlike artificial intelligence programmers who aim to build programs that complete tasks faster and with fewer errors than humans, computational cognitive modellers try to write computer programs that complete tasks in exactly the same way as humans i.e. that make all the mistakes that humans make, and take as much time as humans do to perform tasks.

Computational cognitive models are models which help designers to understand how the human mind works and how users learn and interpret information and how they interact with computers. They are

used in HCI because they help to identify and explain the nature of problems which users encounter, and provide knowledge about what users can and can not be expected to do, as well as helping designers to understand what is going when users use systems. This also benefits interactive systems design because it allows designers to apply the knowledge from these models to build better equipment and interfaces with improved usability and a design which users can understand with a shallower learning curve.

In recent years, cognitive modelling has grown in popularity as a research method in Cognitive Science generally and specifically in the area of HCI research. This can be seen by the increase in the number of published papers reporting results of modelling efforts, the number of text books on the subject (e.g. Boden 1988, Polk & Seifert 2002, Cooper 2002, Gluck & Pew 2005 ) and by the growth of the International Conference on Cognitive Modelling (ICCM) series (e.g. ICCM 2006 http://iccm2006.units.it/index.html ICCM 2007 http://sitemaker.umich.edu/iccm2007.org/home).

There are too many architectures in existence for us to provide a comprehensive review here, therefore the reader is directed to Gray, Young & Kirschenbaum (1997) for a more detailed discussion of cognitive architectures and their use for HCI. As ACT-R is the architecture that has been used most in recent years to build computational models of HCI we will outline its structure here and provide an example of its use in section 4.

### 5.3.1 ACT-R

ACT is a theory of human cognition developed over a period of 30 years by John Anderson and his colleagues that builds on the theory of rational analysis. It is a principal effort in the attempt to develop a unified theory of cognition (Newell, 1990). Over the years, ACT has developed substantially and in doing so, its name has varied to reflect these new developments (ACT*, ACT-R, ACT-R(PM), etc). The current version is known as ACT-R 6.0 and this is the version that is outlined here (Anderson & Liebere 1998).

Figure 5.2 illustrates the components of the architecture relevant to our discussion. ACT-R consists of a set of independent modules that acquire information from the environment, process information and execute motor actions in the furtherance of particular goals. There are four modules that comprise the central cognitive components of ACT-R. Two of these are memory stores for two types of knowledge: a declarative memory module that stores factual knowledge about the domain, and a procedural memory module that stores the system's knowledge about how tasks are performed. The former consists of a network of knowledge chunks whereas the latter is a set of productions, rules of the form "IF <condition> THEN <action>": the condition specifying chunks that must be present for the rule to apply and the action specifying the actions to be taken should this occur. The are a further two cognitive modules (not shown in the diagram) that represent information related to the execution of tasks. The first is a control state module that keeps track of the intentions of the system during problem solving, and the second is a problem state module that maintains the current state of the task.

In addition to these cognitive modules there are four perceptual-motor modules for speech, audition, visual and motor processing (only the latter two are shown in Figure 5.2). The speech and audition modules are the least well-developed and at present simply provide ACT-R with the capacity to simulate basic audio perception and vocal output for the purpose of modelling typical psychology experiments. The visual and motor modules are more well-developed and provide ACT-R with the ability to simulate visual attention shifts to objects on a computer display and manual interactions with a computer keyboard and mouse.

Each of ACT-R's modules has an associated buffer that can hold only one chunk of information from its module at a time and the contents of all of the buffers constitute the state of an ACT-R model at any one time. Cognition proceeds via a pattern matching process that attempts to find production rules with conditions that match the current contents of the buffers. When a match is found, the production "fires" and the actions (visual or manual movements, requests for the retrieval of a knowledge chunk from declarative memory, or modifications to buffers) are performed. Then the matching process continues on the updated

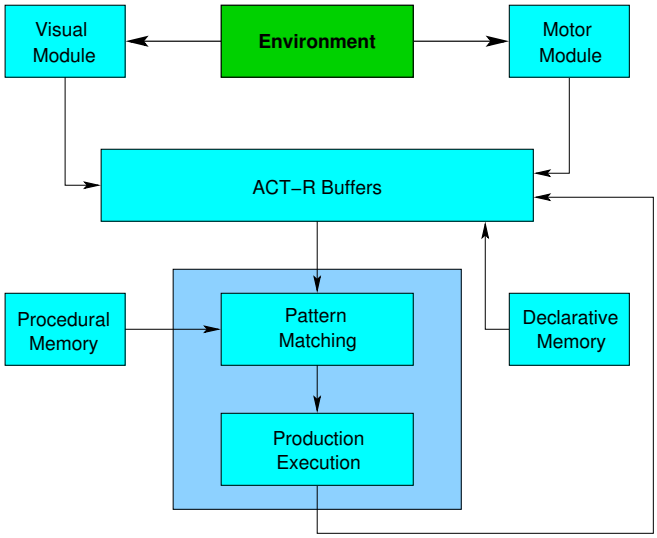


Figure 5.2: The modular structure of ACT-R 6.0

contents of the buffers so that tasks are performed through a succession of production rule firings. As an example, two production rules (written in English rather than in ACT-R code) that instantiate part of a search task may look something like this:

IF the goal is to find the meaning of *eudaimonia* (control state module)
AND there is nothing in declarative memory about *eudaimonia* (declarative module)
THEN set the goal to search the WWW for *eudaimonia* (control state module)

IF the goal is to search the WWW for *eudaimonia* (control state module)
AND the web browser is open (problem state module)
THEN look for the menu labelled *Bookmarks* (visual module)
AND update the problem state to *looking for Google* (problem state module)

The processing in ACT-R's modules is serial but the modules run in parallel with each other so that the system can move visual attention while also moving the mouse and attempting to retrieve knowledge from declarative memory. ACT-R processes also have associated latency parameters taken from the psychology literature. For example, it typically takes 50 ms for a production to fire and the time taken to move the mouse cursor to an object on the computer screen is calculated using Fitts' Law (Fitts, 1954).

ACT-R implements rational analysis in two ways. The first is its mechanism for retrieving knowledge chunks from declarative memory which is based on the notion of activation. Each chunk in declarative memory has a level of activation which determines its availability for retrieval and the level of activation for a chunk reflects the recency and frequency of its use. This enables us to understand how rehearsal of items in a short-term memory task can boost the activation levels of these chunks and consequently increase the chances of recall/retrieval from the declarative memory store. The level of activation of a chunk falls gradually over time and without retrieval or spreading activation from cue chunks may fall below a threshold level which then results in retrieval failure. This enables models built within the ACT-R architecture to model forgetting without having to delete the item from the declarative memory store.

The second way that ACT-R implements rational analysis is in the mechanism for choosing between alternative production rules. According to rational analysis, people choose between a number of options to maximise their expected utility. Each option (production rule) has an expected probability, P, of achieving the goal and an expected cost, C. It is assumed that when carrying out computer-based tasks people

interact with the task environment and choose actions that will optimise their efficiency (i.e. maximise the probability of achieving the goal while minimising the cost). At each point in time therefore all possible production rules that match against the current goal are proposed in a choice set and the one with the highest level of efficiency is chosen and executed.

## 5.4 Applying the method

### 5.4.1 Peebles & Cheng 2003

Modern computer systems (e.g. statistical packages such as SPSS, spreadsheet packages such as Excel, and Geographical Information Systems) are able to manipulate large amounts of data very quickly and produce representations of different aspects of that data. These systems enable users to create different types of graphical representations in order to aid understanding of the data itself. However, little is currently known about how different aspects of the representations can influence people's abilities to extract information from the representations once they have been created. In order to address this, Peebles and Cheng (2003) conducted an experiment, eye movement study and cognitive modelling analysis to investigate the cognitive, perceptual and motor processes involved in a common graph reading task using two different types of Cartesian graph. The purpose of the study was to determine how graph users' ability to retrieve information can be affected by presenting the same information in slightly different types of the same class of diagram. The two types of graph, shown in Figure 5.3, represent amounts of UK oil and gas production over two decades. The only difference between the two graph types is in which variables are represented on the axes and which are plotted. In the Function graphs, the argument variable (AV: time in years) is represented on the x-axis and the quantity variables (QV: oil and gas) on the y-axis whereas in the Parametric graphs, the quantity variables are represented on the x and y axes and time is plotted as a parameterising variable along the curve.

In the experiment, participants were presented with the value of a *given* variable and required use the graph to find the corresponding value of a target variable, for example, when the value of oil is 2, what is the value of gas? This type of task his typically been analysed in terms of a minimum sequence of eye fixations required to reach the location of the given variable's value and then from there to the location of the corresponding value of the target variable (Lohse, 1993; Peebles et al., 1999; Peebles & Cheng, 2001, 2002). Forty-nine participants (four of whom had their eye movements recorded) completed 120 trials, each participant using only one graph type. The 120 questions were coded into three classes (QV-QV, QV-AV and AV-QV) according to which variable's value was given and which was required (QV denotes a quantity variable, oil or gas, and AV denotes the argument variable, time). On each trial, a question (e.g., "GAS = 6, OIL = ?") was presented above the graph and participants were required to read the question, find the answer using the graph on the screen and then enter their answer by clicking on an Answer button on the top right corner of the window which revealed a circle of buttons containing the digits 0 to 9. Reaction times (RTs) were recorded from the onset of a question to the mouse click on the answer button.

The RT data from the experiment, displayed in Figure 5.4, showed that the graph used and the type of question asked both had a significant effect on the time it took for participants to retrieve the answer. This was all the more surprising because, for two of the three question types, participants were faster using the less familiar parametric graphs by nearly a second.

The results of the eye movement study were also surprising. It was found that in 62.7% of all trials (irrespective of the graph used or question type being attempted), after having read the question at the start of a trial, participants redirected their visual attention to elements of the question at least once during the process of problem solving with the graph. This was not predicted by the simple minimal fixation sequence account outlined above but two possible explanations may be provided: (a) participants initially encode the three question elements but are unable to retain all of them in working memory or retrieve

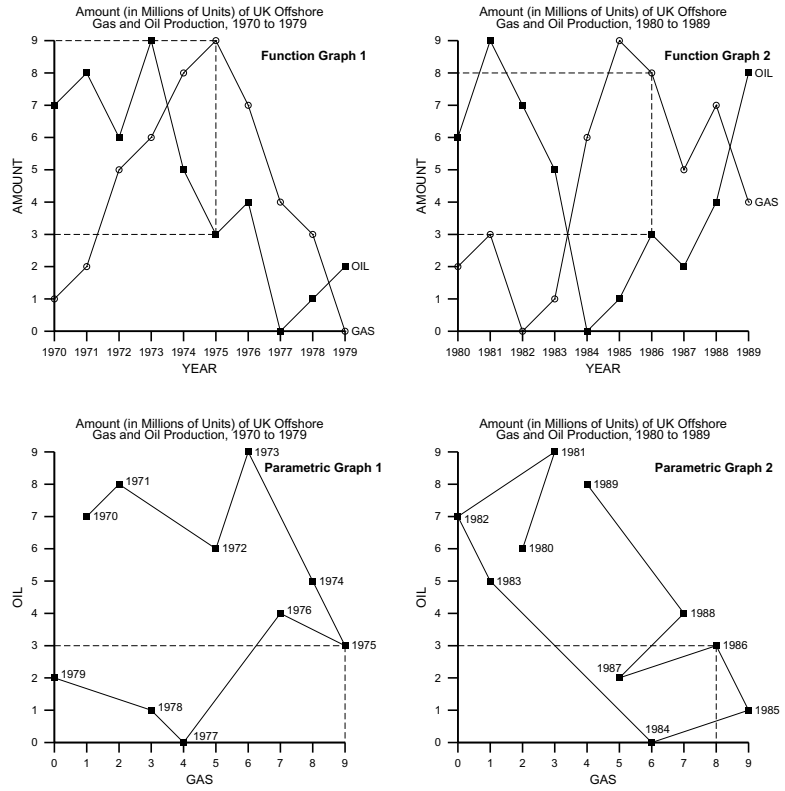


Figure 5.3: Function and Parametric graphs used in Peebles & Cheng (2003) depicting values of oil and gas production for each year. The graphs on the left (labelled 1) show years 1970 to 1979 while those on the right (labelled 2) show years 1980 to 1989. Dashed lines indicate the optimal scan path required to answer the question "When the value of oil is 3, what is the value of gas?"

them when required due to the cognitive load involved in solving the problem, or (b) to reduce the load on working memory, participants break the problem into two sections, the first allowing them to reach the given location and the second to then proceed to the target location corresponding to the solution.

Peebles & Cheng constructed two ACT-R models of the experiment (one for each of graph type) that were able to interact with an exact replica of the software used to run the experiment. The models consisted of a set of production rules to carry out the six basic subgoals in the task; (a) read the question, (b) identify the start location determined by the given variable (c) identify the given location on the graph representing the given value of given variable, (d) from the given location, identify the target location representing the required variable, (e) identify the target value at the target location, (f) enter the answer. Many of the production rules were shared by the two models, the main difference between them being the control structure that sequences the execution of the production rules. Figure 5.4 shows that the mean RTs from the parametric and function graph models are a good fit to the observed data ($R^2 = .868$, RMSE $= 0.123$, and $R^2 = .664$, RMSE $= 0.199$ respectively). Perhaps more importantly however, were the insights into the observed eye movement data that came from the modelling process itself. When ACT-R focuses attention on an object on the screen, representations of the object and its location are created in the system's visual buffers which can be accessed by productions. Eventually these representations go into declarative memory with initial activation values and, as long as these values are above a certain threshold, they can be retrieved by the cognitive system and replaced in a buffer. However, as we mentioned earlier, ACT-R includes a mechanism by which the activation of representations in declarative memory decreases

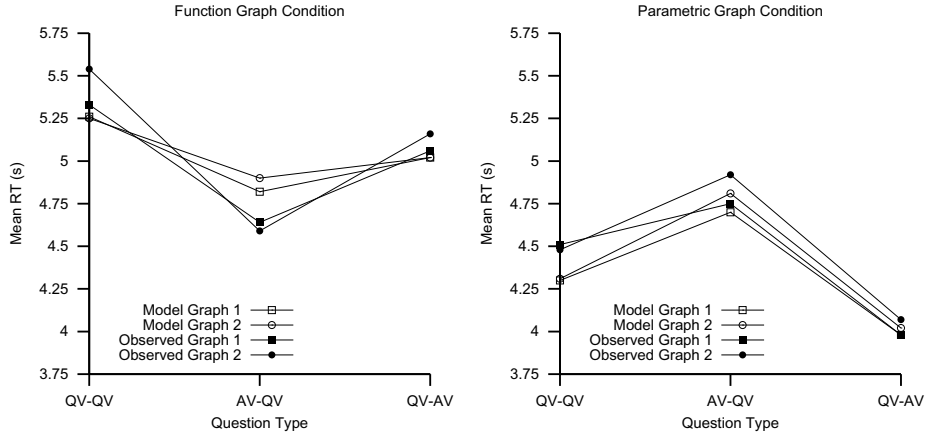


Figure 5.4: Mean response times for experimental participants and ACT-R models for each question type (Peebles & Cheng, 2003).

over time which allows it to simulate processes involved in forgetting. These mechanisms played a crucial role in the ACT-R models' ability to capture the eye movement data observed in the experiment. At the start of each trial, the models read the three question elements but during the processing of the trial these elements do not stay in the visual buffers but are placed in declarative memory. As a consequence, at least one question element must be retrieved from memory at each stage of the problem in order to continue. However, as soon as a question element is placed in declarative memory its activation starts to decay and, as a consequence, the probability that it cannot be retrieved increases. Typically, if a retrieval failure occurs, an ACT-R model will halt as it does not have the appropriate information to solve the problem. During the process of model development it was found that on a significant proportion of trials the model was not able to retrieve question elements at the later stages of the trial because their activation had fallen below the retrieval threshold. As a consequence new productions had to be added to allow the model to redirect attention to the question in order to re-encode the element and then return to solving the problem. This was precisely the behaviour observed in the eye movement study. This is illustrated in Figure 5.5 which compares screen shots of the model scan path and eye movements recorded from one participant for the same question using the 1980's parametric graph. The numbered circles on the model screen shot indicate the sequence of fixations produced by the model. The pattern of fixations in both screenshots is remarkably similar.

### 5.4.2 Salvucci 2001

Most HCI research concentrates on understanding users interacting with one desktop interface. However, this research looks at non-desktop interfaces and in fact includes interaction with more than one interface. The starting point for the research was an existing ACT-R model of driving (steering and speed control) which was further developed so that it included user performance of dialling a telephone number while driving. In building the model, Salvucci decided to compare the effects of dialling a phone number in different modes (full manual dialling, full voice, speed manual, speed voice) on the model's ability to control the car. In order to do this, he took a GOMS description of dialling a telephone number with each method of interaction and expanded it to include cognitive processes. Each of the GOMS operators and cognitive processes included in the task descriptions was taken to represent a production in the model.

This multi-tasking model was used to provide a priori predictions about task performance which

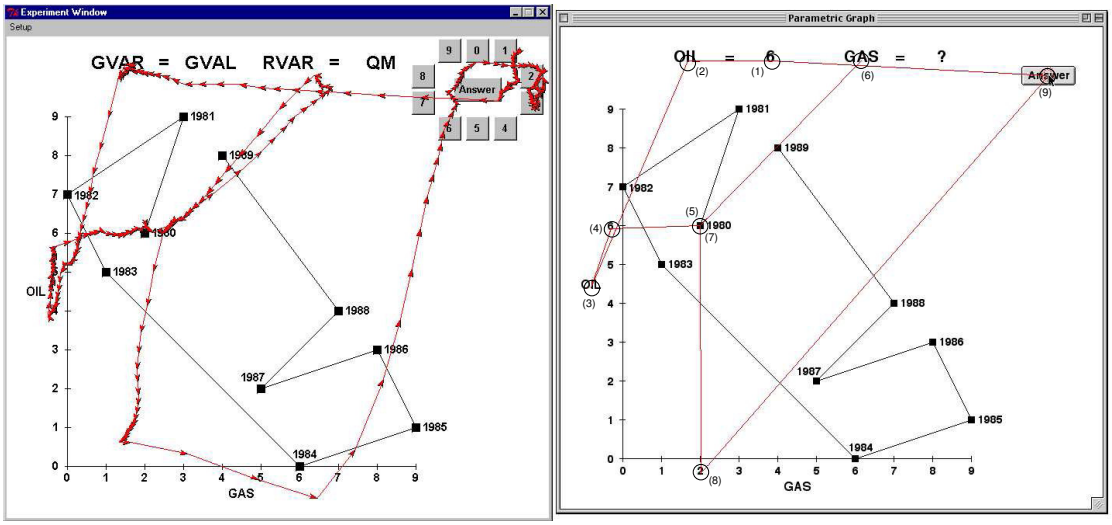


Figure 5.5: Screen shots showing an experimental participant's eye movement data (left) and the ACT-R model's visual attention scan path (right) for the QV–QV question "oil = 6, gas = ?" using the 1980's Parametric graph. In the model screen shot, numbered circles on the scan path indicate the location and sequence of fixations.

were then compared to human data from an experiment. The model made a number of predictions and provided explanations as to why these predictions were sensible. For example, perhaps unsurprisingly, the model predicted that the full-manual telephone interface has a large effect on driver behaviour but importantly using the speed-manual interface (i.e. using a single digit speed dial number) also has a small but significant effect. The models also showed that use of the voice interfaces has no effect on driving performance. This was because both manual interfaces occupy more visual attention than the voice interfaces as the model needs to look at the keypad while dialling. The voice interfaces require aural rather than visual attention which only detracts a small amount from the model's attention to the driving task. As suggested in the cognitive psychology literature the model can more easily switch between the different types of attention (visual for driving, aural for dialling) than switch visual attention between the two tasks when using the manual telephone interfaces.

The main benefit of the model is its predictive power. The models closely predict the baseline dialling times for the four different interfaces when not driving. Coupled with the driving task, the models have the ability to predict interactions between task behaviours which provides us with a real understanding of not just which multi-tasking combinations are easier to use but also why this is the case. In addition, if we created a new form of interaction for this task, we could just build another user model and plug it in to the driving model, enabling us to look at that multi-tasking situation. It would also be possible to change driving conditions to look at whether there are some situations where interacting with the telephone interfaces might be more or less dangerous. Would it be safer to use the manual interface on a single lane road with no other traffic (as in Salvlucci's experiment), on a single lane road with other traffic or on a motorway which is likely to be straighter, with no oncoming traffic to contend with?

## 5.5 Critique

The Peebles and Cheng (2003) case above provides a good example of some of the benefits of cognitive modelling. First, engagement in the activity of modelling an aspect of human behaviour helps to clarify

the theory about the cognitive processes involved in completing the task of interest. Peebles and Cheng were forced to be explicit about the cognitive strategies they thought humans employed when trying to understand the data presented in the graphs. Second, the behaviour of the resulting model can be compared to that of human participants to ensure that the account of the processes involved is accurate. It was in doing this that Peebles and Cheng realised that they needed to extend their models in order to capture the redirection of attention to the task description before the model could complete the task.

Cognitive models can also be used to make predictions about human behaviour in other situations (such as interacting with other types of graphs), and be used to compare systems and/or competing designs as part of the usability evaluation process. Computational cognitive modelling is of particular value to HCI researchers as models built within a computational environment can be hooked up directly to the software applications that human users use. More recent architectures provide the researcher with the ability to re-use all or parts of previous models, and even to combine two or more existing models in order to investigate how multi-tasking might impact performance (see summary of Salvucci 2001 above).

### 5.5.1 What to consider when choosing a method for your own research

Having provided the reader with a description of the benefits of cognitive modelling in HCI, in order for us to give a balanced view it is necessary to consider some of the limitations of the approach because building cognitive models of human-computer interaction is not an easy or trivial matter.

First, it is important to spend some time deciding which model or architecture is most appropriate for the task or data set that you are trying to model. For a detailed discussion of how to choose and get started with a cognitive architecture, the reader is directed to Ritter (2004). In practice, unless you are a programmer who is already proficient in the language that underpins the cognitive architecture, the learning curve required to build a computational model is too steep for someone who is undertaking a 3 month MSc project. So, in this circumstance you are probably better off limiting yourself to GOMS style models. For many system evaluation type projects, using a member of the GOMS family of models is likely to be sufficient. Those involved in a larger project, or at least one that will run over a longer time period, such as a PhD thesis, are more likely to have the time required to learn both the underlying language and how to build a model and will be able to take advantage of resources such as the ACT-R summer school. Tools are already being developed to aid the modeller to automatically generate GOMS (e.g. Vera et al 2005) and ACT-R models (e.g. CogTool by Salvucci & Lee (2003), ACT-stitch by Matessa (2004)). These will facilitate the move from models of expert behaviour with an interface (in GOMS) to models of novice behaviour (in ACT-R).

Once you have built your model it is necessary to test it against some human data. Many papers report the results of such efforts claiming that because their model behaves in some way similar to their participants this in fact validates their model. However, it is much easier to build a model that behaves in a particular way if we already know how it should behave i.e. if we have already collected and analysed the human data. Once the modeller has tweaked with some of the input parameters, it is perhaps of little surprise that he is able to get the model to match the human data. The cleaner approach is therefore to build the model before examining the human data. If the behaviour of the model does not match the human data then, like Peebles & Cheng, it is important to consider where the model might need to be changed, rather than trying to retrofit the model to the data via parameter fitting.

A final point to remember is that a cognitive model is a model of human-computer interaction in an ideal environment where there are no other factors, such as interruptions or emotional issues, which might affect thought processes. As such, predictions made by the model cannot always be taken as absolute truth. This thought may go some way to lessening the blow when, despite your best efforts, you find it difficult if not impossible to accurately model your data.

## 5.6 Sources for more information

ACT-R web site includes publications, tutorial material, etc.
http://act-r.psy.cmu.edu/

Cooper, R. P. (2002): Modelling High-Level Cognitive Processes. Lawrence Erlbaum Associates:

Gluck, K.A. & Pew, R.W. (Eds) (2005), Modeling Human Behaviour With Integrated Cognitive Architectures.

Gray, W. D., Young, R. M., & Kirschenbaum, S. S. (1997). Introduction to this Special Issue on Cognitive Architectures and Human-Computer Interaction. Human-Computer Interaction, 12(4), 301?309.

Kieras, D.E. (2004) EPIC Architecture Principles of Operation.
ftp://www.eecs.umich.edu/people/kieras/EPICtutorial/EPICPrinOp.pdf

Ritter, F. E. (2004). Choosing and getting started with a cognitive architecture to test and use human-machine interfaces. MMI-Interaktiv-Journal's special issue on Modeling and Simulation in Human-Machine Systems. 7. 17-37.

Polk, T. & Seifert, C. (2002). *Cognitive Modeling*. MIT Press