On Emotion, Learning and Uncertainty: A Cognitive Modelling Approach

by Roman V Belavkin, MSc

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy, August 2002

Contents

Abstract

A problem of emotion and cognition is considered within a unified theory of cognition. There is a strong case for modern cognitive models to take arousal component of emotion into account because of its significant influence on performance (e.g. the inverted–U effect). Several hypotheses have been proposed to explain this effect, but they have not been integrated into cognitive architectures. Based on the analysis of the ACT–R (Anderson & Lebiere, 1998) cognitive architecture the mechanisms that can be used to model this effect are identified. Then a model of the classical Yerkes and Dodson experiment is introduced. The model matches the data by modifying several parameters, particularly noise and goal value in the conflict resolution strategy. Thus, the model supports the idea that the character of decision making changes for different arousal and motivational states. The effect of these changes on learning is analysed using information theory. In particular, randomness in behaviour due to a noise increase leads to a faster entropy reduction. Thus, noise can improve learning in the initial stage of problem exploration or upon changes in the environment. Furthermore, dynamic motivation can optimise the expenditure of effort. Therefore, emotion may play an important role in adaptation of cognitive processes. It is argued that the current conflict resolution mechanism in ACT–R does not explain the dynamics suggested by the model. A new theory and algorithm are proposed that use posterior estimation of expected costs. There are three main contributions of the thesis: 1) Ways of including the effects of emotion and motivation into cognitive models; 2) The analysis of the role of emotion in learning and intelligence; and 3) The introduction of a new machine learning algorithm suitable for applications not only in cognitive modelling, but in other areas of computer science.

Acknowledgements

I would like to thank Frank Ritter, David Elliman and David Wood for providing brilliant support, inspirational discussions, and patient reading of drafts during the supervision of this thesis. I also wish to thank my father, Viacheslav Belavkin, for clarifying some of the mathematical issues. This work was funded by the ESRC Centre for Research in Development, Instruction, and Training (CREDIT), the University of Nottingham, and by an Overseas Research Studentship from the Committee of Vice Chancellors and Principles (CVCP), United Kingdom.

To Rachel and her dog Didi and also to my parents.

CHAPTER 1

Introduction

'*A test of true thinking must involve emotion*' (Picard, 1997, page 12).

The subject of emotion has puzzled many psychologists, philosophers and neurobiologists since William James first attempted to define emotion (James, 1884). Many theories of emotion, sometimes quite contradictory, have appeared since then (see Plutchik, 1994; LeDoux, 1996; Lambie & Marcel, 2002, for reviews). Recently the subject of emotion has attracted the attention of the computer science and artificial intelligence communities, and has emerged into a new area of research, sometimes referred to as *affective computing* (Picard, 1997).

Although without a doubt emotion is a very important component of human and animal psychology, one of the most intriguing and interesting question remains unanswered: Is emotion a necessary component of intelligence? And if it is, how should it be included into the AI theory?

1.1 Emotion and Intelligence

The idea that emotion is important for intelligence is not new, and the term *emotional intelligence*, introduced by Salovey and Mayer (1990), has been the subject of a popular book by the philosopher Daniel Goleman (1995). It is known from experimental psychology that emotion is closely related to many cognitive processes, particularly decision making (Tversky & Kahneman, 1981; Johnson & Tversky, 1983), memory (Brown & Kulik, 1977), and perception. Recently there have been claims, based on some experimental evidence (Damasio, 1994), that damage to the brain limiting the influence of emotion impairs intelligence. Particularly, Damasio studied patients who due to some accidents and operations

had restricted connections in the brain between the frontal lobe, normally associated with the executive functions and control, and the areas in the autonomic nervous system (ANS), which is believed to be responsible for emotions. These patients, as described by Damasio in his book, although apparently fully recovered after their operations, displayed remarkable abnormalities in their behaviour. In particular, they were very indecisive when it came to choosing between many similar options, and they had a tendency to repeat the same errors many times. Thus, it was claimed, the lack of emotion impaired their decision making and learning mechanisms. However, as noted by Sloman (1999), these claims are a little premature, as the damaged areas of the brain are not yet sufficiently understood.

1.2 The Need to Include Emotion in Cognitive Models

Cognitive modelling is the area of cognitive science in which theories of cognition are evaluated by comparing the performance of computer simulations with data from psychological experiments. Recent progress in cognitive modelling has facilitated the testing of a broad range of psychological and cognitive theories. Extensive work in experimental psychology has been reproduced and reconsidered by cognitive modellers within the unified theories of cognition and their implementations, such as SOAR (Newell, 1990) and ACT–R (Anderson & Lebiere, 1998). This work has produced new insights into our understanding of perception, memory, learning, decision making. The knowledge acquired has been invaluable for advancing artificial intelligence research. There have been few attempts, however, to study emotion within the cognitive modelling framework, even though the need to incorporate emotions into the cognitive theory was pointed out early on by Herbert Simon (1967).

On the other hand, it is becoming evident that cognitive models have approached the stage where the effect of emotion should be taken into account (Belavkin, Ritter, & Elliman, 1999). Recently some researchers have been successful in simulating quite complex tasks, such as solving various non–trivial puzzles (Anderson, Kushmerick, & Lebiere, 1993; Jones, Ritter, & Wood, 2000), playing chess (Gobet & Jansen, 1994), the interaction with graphical user interfaces (Ritter, Baxter, Jones, & Young, 2000; Fleetwood & Byrne, 2001), airtraffic control (Taatgen, 2001), and even the navigation of vehicles and aeroplanes (Salvucci & Macuga, 2001; Schoppek, Holt, Diez, & Boehm-Davis, 2001). The performance of subjects in these tasks can be greatly influenced by emotional experiences. Furthermore, some models, such as the Tower of Nottingham model (Jones et al., 2000), consider child

subjects, and their emotions are easily observable. Yet few of these computer simulations say much about the emotion. It seems that apart from a few studies (e.g. Ritter, 1993) in cognitive science and modelling the subject of emotion has been neglected.

1.3 Review of Emotion Research

Perhaps the biggest challenge in emotion research is the fact that there is no unified theory of emotion. Psychologists and philosophers still cannot agree on the fundamental point in the subject about what comes first: feeling or thought. Do we feel afraid because we run from a bear (James, 1884), or do we run from the bear because we are afraid (Cannon, 1915, 1929; Bard, 1934)? This debate has been transformed more recently into the discussion between the followers of the cognitive appraisal traditions of Schachter and Singer (1962), in which cognition is considered to be the main elicitor of emotions, and their critics (e.g. Zajonc, 1980; LeDoux, 1990). The latter favour the idea that emotions arise from subconscious appraisal processes and may occur independently of any conscious awareness of the reasons causing the emotion (unlike cognitive appraisal). The main argument against cognitive appraisal was the fact that people sometimes can feel sad without clearly understanding the reasons why. In addition, the results of some more recent experiments suggest that feelings associated with at least primary emotions (e.g. fear) can occur ahead of their conscious awareness (LeDoux, 1990). Also, some stroke patients described by Damasio (1994) reported that they no longer felt sadness or happiness in certain scenes of movies they had previously watched, although their memories and experiences suggested that they should have. Thus, reasoning about emotion can occur without the emotional experience itself.

Despite the great controversy of different theories a number of computational models have been developed (see Hudlicka & Fellous, 1996; Picard, 1997, for reviews). For example, a very influential model of cognitive appraisal have been proposed by Ortony, Clore, and Collins (1988). This model allows the mapping of external stimuli and agent's internal representations onto a set of emotions. There are other models of cognitive appraisal, such as the rule–based model by Scherer (1993), which allow reasoning about emotions based on the subject's motivations, environmental factors, etc. Another example is the categorisation model by Roseman, Antoniou, and Jose (1996).

Cognitive appraisal models are used in agent architectures to express emotions by

intelligent agents and robots (e.g. Bates, Loyall, & Reilly, 1992; Cahn, 1990; Scheutz & Logan, 2001). Although these models can correctly predict which emotions a subject would feel, unfortunately, these symbolic representations do not explain what happens to the thinking process itself as a result of these emotions. For example, what are the implications of assembling the Tower of Hanoi in an angry or a happy mood? It is clear that the cognitive appraisal models alone cannot explain the contribution of emotion to intelligence.

An important development in this area became the work by Frijda (1986) who proposed a theory in which emotions are seen as a control system of an agent. In this theory and its computational implementation (Frijda & Swagerman, 1987) emotions monitor the goals satisfaction and may change the behaviour of an agent accordingly by sending interrupt signals. A similar approach has been adopted by Oatley and Johnson-Laird (1987).

A different approach to understanding emotion and its relation to cognition was suggested by Aaron Sloman and his group (Sloman, 2001). They see emotions, as well as other psychological phenomena, as a consequence of the architectural requirements of different animals or agents, which is necessary to survive or achieve their goals. By defining more sophisticated requirements for agent architectures they look into what states these architectures support. It is claimed that behaviour similar to emotional reactions could be observed in these agents. For example, some regulatory mechanisms can put an agent into different modes depending on the situation in the environment or the agent itself, similar to the states associated with primary emotions such as fear or anxiety. Unfortunately the lack of reference to psychological theories does not allow their architectures to be tested against experimental data.

Some of the more psychologically grounded research is the work of Dörner and colleagues. Their PSI–theory attempts to integrate cognition, emotion and motivation using connectionist approach (Dörner & Hille, 1995; Bartl & Dörner, 1998). PSI was implemented as a computational model and tested onl data from subjects. Some of the important features of this work are emotional modulations of information processing and action control. In addition, PSI is able to express its emotions using a picture of a human face. The emotions observable in PSI's facial expression and behaviour resembled the expressions of the subjects. PSI is remarkable because it is one of the few attempts to compare emotions expressed by a model and their effect on behaviour with data. This approach of testing theoretical predictions by grounding them in experimental data will be adopted throughout this thesis.

1.4 Data Relating Intelligence and Emotion

As was mentioned earlier, emotion is believed to be involved in cognitive processes crucial for intelligence, such as motivation, decision making and memory. One may wonder if there exists any experimental data supporting these claims, and which can be tested on a model. Some of the examples of such experiments can be found in the studies of the *inverted–U* effect (see Easterbrook, 1959; Anderson, 1990, for reviews).

The first experiment in which the inverted–U effect was observed is the famous 'dancing mouse' experiment of Yerkes and Dodson (1908). In this experiment they studied the speed of learning in mice under different levels of stimulation and task difficulty. The mice were placed into a discrimination chamber with two exits, and they were trained for several days always to escape through one particular door marked by a white card. If a mouse tried to escape through a door with the black card, it received an electrical shock. The order of the doors was changed randomly, so the mice had to learn to choose the door based on its colour rather than on its position. They studied learning in several groups, each trained using different strength of electrical signals and under different conditions of visual discrimination. The main result was that mice trained with a medium signal learned faster than those trained with either a weak or a strong stimulus. Thus, it was found that performance (speed of learning) increases only up to a certain level of stimulation, and stronger signals hinder the performance. This effect was especially noticeable if the task was more difficult (i.e. in darker conditions with poorer discrimination).

A series of other experiments demonstrated the inverted–U effect in various tasks (e.g. Näätänen, 1973; Gupta, 1977; Anderson & Revelle, 1982), which led to the inverted–U hypothesis that cognitive performance is curvilinearly related to the level of *arousal* (see Figure 1.1). Arousal is a generalised term describing different levels of activation of the ANS, such as fatigue or alertness, and it is related to the levels of external or internal stimulation, such as sensory inputs, emotion, etc (for discussion of the term arousal see Hebb, 1955; Thayer, 1978; Humphreys & Revelle, 1984; Anderson, 1990). The level of arousal can be observed experimentally, for example using GSR (*galvanic skin response*) or EDR (*electrodermal response*).

The multidimensional nature of arousal helps to explain the inverted–U phenomenon. For example, Humphreys and Revelle (1984) suggested that different cognitive parameters ('memory capacity', 'information transfer', etc) depend differently on arousal,

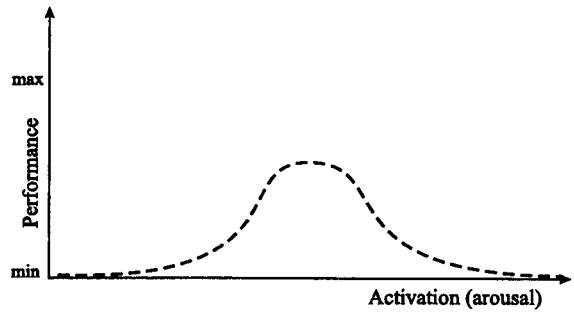

FIGURE 1.1: The inverted–U hypothesis relating cognitive performance to the level of arousal.

and, as a result, performance is a curvilinear (inverted–U) function of arousal with maximum at a particular point, which may also depend on a task and individual differences.

Although the inverted–U hypothesis as well as the term arousal itself have been periodically criticised (Broen & Storms, 1961; Neiss, 1990), there exists a wide range of experimental data supporting the phenomenon (see Anderson, 1990, for a review). In particular, a number of experiments was devoted specifically to the relation between performance and arousal associated with strong emotions such as anxiety (Mandler & Sarason, 1952; Liebert & Morris, 1967; Wine, 1971; Matthews, 1985), and their results are in favour of the Yerkes and Dodson law.

The value of the inverted–U phenomenon and experiments for this research is that they provide data relating the two crucial concepts: performance, which is related to intelligence, and arousal, which is an important characteristic of emotional experience.

1.5 Objectives

In this research the subject of emotion will be considered from the practical point of the need to include the effect of emotion on problem solving in cognitive models. Avoiding the controversy of many different theories, definitions of emotions and related philosophical questions we shall start from modelling a particular classical experiment — the Yerkes and Dodson inverted–U experiment. The reason why this experiment has been chosen is, perhaps, its importance and popularity. Indeed, despite its significant age the article is referenced practically in every work related to arousal and the inverted–U effect (the 'Web of Science' web–site found over 700 citations). In addition, the task itself is not very difficult:

the model has to learn which of the two doors to choose. A cognitive model that predicts the results of such an experiment may help us to understand the functional relation between cognition and the principal components of emotions (i.e. arousal).

The objectives of the study are listed below:

1. Analyse the state of the art in cognitive architecture and the results of some models in order to determine the methods suitable for creating the model.
2. Create a model of the Yerkes and Dodson experiment which is sufficiently complex to obtain a fair match with the data.
3. Analyse the model results and articulate its implications for the nature of the effects of arousal on cognitive processes.
4. Test whether there is any consistent discrepancy due to the limitations of the current theory. Refer to other cognitive modelling work to see whether similar problems have been observed.
5. If such discrepancies are found, propose methods to improve cognitive models' performance on tasks in which the effects of emotions are important.
6. Develop a new theory that could incorporate the suggested changes.
7. Analyse the value of this work and its implications for other areas of AI and computer science.

1.6 Overview of the Thesis

In the next chapter we shall briefly describe and analyse ACT–R (Anderson & Lebiere, 1998) — currently the most popular cognitive architecture.[1] Then some interesting results of existing ACT–R models and related problems will be outlined. The properties of some variables and mechanisms in ACT–R, particularly the decision making, learning and memory mechanisms, will be analysed, and a method to model the inverted–U effect by modifying parameters of these mechanisms will be proposed.

The model of the Yerkes and Dodson experiment will be described in Chapter 3, and its complete code in Appendix A. The results of several modifications of the model

[1] The majority of ACT–R models presented for the *Fourth International Conference on Cognitive Modelling 2001* was overwhelming.

will be presented in Chapter 4. It will be shown that performance under different levels of stimulation (and arousal) corresponds to different values of motivational and randomness determinants of the ACT–R decision making mechanism. The possible implications of arousal on memory will be also considered.

Chapter 5 will consider the implications of different modes of decision making on the effectiveness of learning. By using the information theory approach it will be shown how randomness in behaviour controlled through the noise variance in ACT–R helps the model to learn faster at certain stages of problem solving. It will be suggested that the level of randomness (noise) reflects the level of uncertainty about the task. It will be shown that the level of noise controlled dynamically by entropy parameter can significantly improve the match between model and data. The dynamics of the decision making parameters will be compared with some known search and optimisation techniques. It will be suggested that emotion and arousal shift the decision making strategy dynamically in a similar way to the heuristic methods already known in AI. Thus, emotion may indeed make problem solving more adaptive and improve intelligence.

The current ACT–R theory does not include the proposed dynamics. A new algorithm for decision making and learning will be described in Chapter 6. This new method possesses the dynamics of motivation and randomness, which is controlled automatically through the interaction with the environment. In addition, the proposed algorithm allows us to reduce the number of variables in ACT–R. An implementation of the new algorithm for ACT–R will be presented, and the model using this new algorithm will be compared with the data. Some animal learning and neural theories will be presented to support the plausibility of the new method. The similarities and differences of the algorithm to various modern optimisation techniques will be outlined. The self–adaptiveness and low computational cost of the new method suggest that it can be employed for problems outside the cognitive modelling domain, particularly, for search and optimisation problems with unknown solutions distributions.

The discussion of the results and conclusions of the thesis will be presented in the final chapter.

CHAPTER 2

Analysis of ACT Theory and ACT–R Architecture

In this chapter we shall briefly outline the ACT–R cognitive architecture (Anderson & Lebiere, 1998) and the underlying theory. Allen Newell pointed out in his book that ACT–R was perhaps the most complete theory and suggested it (along with his own architecture SOAR) as one of the best candidates for the unified theory of cognition (Newell, 1990). At the moment of writing this thesis ACT–R became the most popular and widely accepted cognitive architecture. In this chapter we shall describe the main equations and mechanisms of ACT–R, which are particularly important for this study. Some interesting results of recent cognitive models implemented in ACT–R will be presented. In particular, the Tower of Nottingham model (Jones et al., 2000), which indicated that the role of noise in ACT–R conflict resolution should be studied more thoroughly. The influence of the ACT–R conflict resolution parameters on the character of decision making will be investigated on examples and using asymptotic analysis. The described properties of ACT–R will be compared with theories of human decision making, motivation and some proposed explanations of the inverted–U effect. Finally, a way to model these phenomena using ACT–R will be suggested in the end of the chapter.

2.1 Symbolic Processing

On the surface of ACT–R lays a symbolic goal–directed production system. New goals are put on the goal–stack and compared with the IF part of production rules, of which one is selected and fired. ACT–R follows the assumption that there are two types of knowledge — declarative and procedural. Therefore, symbols (memory units) in ACT–R are classified into two categories.

2.1.1 Declarative Memory

Declarative knowledge represents objects or facts, which we acquire from the environment. The corresponding symbols in ACT–R are called *chunks* and they form the declarative memory. Altogether chunks represent ACT–R's current knowledge of the problem it is working on and the environment. For instance, Figure 2.1 illustrates schematically a chunk that represents the fact that 3 plus 4 is 7. Another example could be a fact 'fish is an animal that lives in water'.

Chunks in ACT–R can be further categorised by their *chunk–type*. These types represent templates for particular types of knowledge facts. We could compare the chunk–types with classes, and chunks with objects (instances of classes).

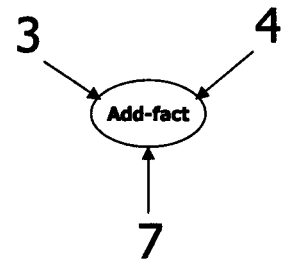

FIGURE 2.1: A chunk representing addition fact $3 + 4 = 7$.

2.1.2 Procedural Memory

Procedural knowledge represents the skills needed to manipulate the declarative facts in order to solve a problem. One may think of the procedural knowledge representing the sub–conscious processes of modification of the working memory contents (i.e. setting a new goal) or actions in the outside world.

Procedural knowledge is encoded in ACT–R in the form of *production rules*. The left–hand side of a rule (IF part or antecedent) must contain a *goal* (one of the chunks) and possibly several constraints (also chunks) to be matched against. The right–hand side (THEN part or consequent) can perform several actions, such as modify the goal, create a new goal, create several new chunks, remove the current goal, etc.

ACT–R matches production rules by the type of a goal–chunk (not by its contents). For example, if the goal is to find the 'sum of 3 and 4', ACT–R looks for a rule with an

addition chunk–type as a goal. An example of the corresponding rule is shown schematically on Figure 2.2. The rule also looks for a constraint chunk representing the answer to the problem — the fact that $3+4=7$. If such fact is found in the memory, then the goal–chunk is modified by filling in the answer.

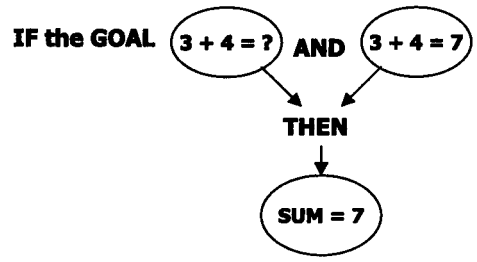

FIGURE 2.2: Schematic representation of a production rule: IF the goal is to find the sum $3+4$ and there is known addition FACT $3+4=7$, THEN the answer is 7.

One might think that the above described process is a bit confusing, because the answer was already known. However, experiments show that we usually do not have to recalculate the sum $3+4$ every time we need the answer. Instead, we quickly recall the answer because we have dealt with this problem many times before. If the answer was not known, then ACT–R model would need to use other production rules to calculate the required result. For example, adding 1 several times: $3+1=4$, $4+1=5$, etc. The problems of cognitive arithmetics have been modelled using ACT–R by Lebiere and Anderson (1998).

2.2 Subsymbolic Processing

The *subsymbolic* level of ACT–R is, perhaps, its main distinguishing feature from other cognitive architectures. In brief, all the symbols in ACT–R (chunks and production rules) have values attached to them (activations, utilities, etc). Moreover, the symbols can be connected by a web of associations, which also have values. These values are controlled by the *rational analysis* mechanism, which acts beneath the symbolic level of the production system. This mechanism can be seen as a type of statistical optimisation (e.g. Bayesian networks). The subsymbolic values can be learned from experience, and may also depend on time. Activations and associations affect many processes in the production system, such as choice of production rules, retrieval of knowledge facts, retrieval times (latencies), etc.

Subsymbolic computations can be controlled through many global parameters,

such as noise variance, goal value, retrieval and utility thresholds and so on. These parameters can dramatically alter the behaviour of ACT–R models. Many of these parameters are used by cognitive modellers to adjust the performance of their models and test theories (e.g. Jones et al., 2000; Lovett, Daily, & Reder, 2000).

One may think that subsymbolic computations is a step back from the pure symbolic production system approach in a sense defined by Post (1943) and used by Newell and Simon (1972). SOAR, for example, does not use subsymbolic computations. Although it is theoretically possible to realise the effects of the subsymbolic computations using many production rules, the benefits of the automatic, parameter–controlled subsymbolic mechanisms are enormous and save a lot of resources for model testing and verification. Moreover, the theory behind the subsymbolic computations of ACT–R has a long history and was inspired by works in cognitive psychology and neuroscience. For example, many constraints implemented in ACT–R Version 4 were derived from the neural implementation of ACT–R — the ACT–RN (Lebiere & Anderson, 1993).

2.2.1 Conflict Resolution

Conflict resolution is a process of selecting one rule out of several matching the current goal state. Many production systems realise different algorithms to overcome such conflicts. ACT–R uses a powerful statistical method that arises from series of studies of subjects' choice behaviour.

In ACT–R each production rule has an *expected gain* E value attached to it (also sometimes called *utility* U). In order to resolve the conflict, ACT–R compares the expected gains of the rules in the conflict set and selects the one with the highest gain. The expected gain is defined as:

$$E_i = P_i G - C_i + \xi(s) \ . \tag{2.1}$$

Here P_i is called the *expected probability* of rule i, and it represents the probability that the goal will be achieved if the rule fires, G is the *goal value* (usually measured in time units), C_i is the *expected cost* representing the effort (also in time units) required to achieve the goal if rule i fires, and $\xi(s)$ is called the *expected gain noise*. The noise here is some random value added to each rule in a conflict set on every cycle. This random value ξ is calculated for each rule individually from logistic distribution with the mean value of zero, and variance

determined by the parameter s:

$$\sigma^2 = \frac{\pi^2 s^2}{3}.$$

Thus, equation (2.1) describes a distribution of E with the mean value $PG - C$ and variance controlled by the parameter s. Noise adds a nondeterministic flavour to the conflict resolution in ACT–R, because when $s > 0$ all production rules even with the same value of $PG - C$ will have slightly different expected gains E.

As one can see, expected probability P_i and cost C_i in (2.1) are properties of a production rule, while goal value G and noise s are global parameters.

The conflict resolution using expected gains (2.1) enabled the modelling of many characteristics of human problem solving. For example, ACT–R models can predict the results of probability matching experiments (Friedman et al., 1964), in which subjects displayed that their choice depends on the rate of successes. Friedman et al. showed, that although subjects choose according to the probability of success (or reinforcement), the proportion of choice of an alternative with the maximum success probability ($P = 1$) never reaches 1. As one can see from (2.1), expected gain E of a rule increases linearly with the probability P. However, the addition of noise increases the chance of rules with smaller probabilities to win the competition, which corresponds to the opportunistic choice behaviour of humans and animals.

In addition to the probability of success, Myers, Fort, Katz, and Suydam (1963) showed that the choice depends also on the value of a reward (pay–off). This property is reflected in ACT–R in G, the goal value parameter in equation (2.1), which can shift the choice towards the rules with higher probabilities. It has been shown (see Figure 2.3) that ACT–R can successfully match the results of these probability matching experiments (Anderson, 1990; Anderson et al., 1993).

2.2.2 Chunk Activation

Chunks (declarative memory units) have *activation* values defined as:

$$A_i = B_i + \sum_j W_j S_{ji}, \tag{2.2}$$

where B_i is the *base–level* of activation of chunk i, W_j is the *source activation* from chunk j, and S_{ji} is the *strength of association* between chunk i and chunk j.

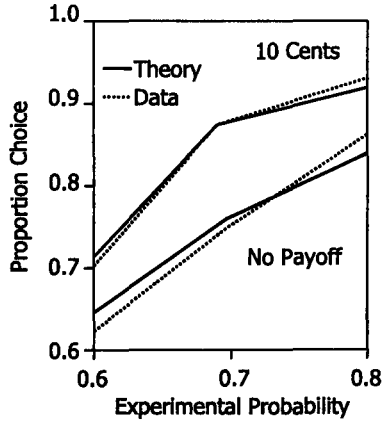

FIGURE 2.3: Proportion of choice as a function of different payoffs and probability (from Anderson & Lebiere, 1998, page 69). The model reproduces data from Myers et al. (1963).

The idea behind W_j and S_{ji} is as follows. Activation of the goal chunk (the source of activation) is spread evenly between the other chunks matched by the production rule (constraints), such that each jth chunk in the production left–hand side receives its part of the goal activation W_j. This part is then multiplied by S_{ji}, which represents how useful chunk i was in the past whenever j was the goal. In fact, S_{ji} is an estimate of a logarithm of the probability that i will be successfully retrieved (see Anderson, Lebiere, & Lovett, 1998, for further details). The products $W_j S_{ji}$ are then summed over all chunks j which referred to i. So, the more productions retrieve a chunk, the higher its activation becomes.

The stochastic and time–dependent nature of memory is reflected in ACT–R in B_i. Base level is a function of time:

$$B(t) = \beta - d \ln t + \xi_1(s_1) + \xi_2(s_2)\,, \tag{2.3}$$

where $\beta = B(0)$ is the *base level constant* (activation of a chunk when it is created), $-d \ln t$ reflects the decay of the base level in time t, where d controls the decay rate, $\xi_1(s_1)$ is the *permanent activation noise* added only at creation time, and $\xi_2(s_2)$ is the *activation noise* added at every cycle. The parameters s_1 and s_2 control the variance of the corresponding noise.

Activations of chunks influence the times needed to retrieve them. One can see from equation (2.3) that because the base–level decays in time, the facts acquired earlier are harder to recall. However, equation (2.3) reflects only part of the dynamics of the activation in time. The base–level learning mechanism of ACT–R reinforces the activation of a chunk

whenever it is used by production rules. This mechanism will be described later in this chapter.

2.2.3 Production Strength

Similar to chunks, productions in ACT–R have their own activations S_p called *production strength*. As with activations affecting the times to retrieve chunks, production strengths affect the time needed to retrieve production rules. However, current versions of ACT–R do not provide as many parameters to control the strength as for the chunks' activations. Particularly, there is no base level constant β, nor is there any controlled noise ξ added to the strengths of productions.

Production strength decays in time similarly to chunk activation. The decay rate is controlled by a separate d parameter. The mechanism of learning production strength is also similar to the learning of base–levels and will be described later.

The production strength mechanism has been criticised recently (ACT–R workshop, 2001) for its inconsistency with the conflict resolution. Indeed, productions have two completely independent values: expected gain E and production strength S_p. One of the main reasons for keeping production strength is that expected gain (2.1) does not reflect which of the productions were learned earlier or later, nor how recently a production has been used. At the ACT–R workshop in 2001 Neils Taatgen proposed an alternative to production strength in a form of time–decaying expected gain noise for each production (e.g. replacing the global noise $\xi(s)$ in (2.1) by $\xi\left(s_i(t)\right)$). The nature of the noise decay, however, was not explained. In this work noise decay will be explained by uncertainty reduction (see Chapter 5). A new conflict resolution algorithm described in Chapter 6 naturally implements the rule–based decaying noise, and can potentially address the issues of production strength.

2.2.4 Retrieval Times

Time periods between the decisions are probably some of the most often studied variables in psychological experiments. A great deal of ACT–R theory is dedicated to *retrieval times* (or *latencies*), which describe the simulated time periods between the cycles (firing of production rules). The success of ACT–R in cognitive modelling is partly explained by its elaborate theory of activation–based retrieval, which provides good predictions of the timing data.

The time needed to retrieve chunk i in production rule p is defined as:

$$t_{ip} = F\, e^{-f(A_i+S_p)}, \tag{2.4}$$

where F and f are scaling factors (by default $F = 1$ sec, $f = 1$), A_i is the activation of chunk i, and S_p is the strength of production p.[1] One can see that times to retrieve chunks exponentially increase for lower activations and production strengths. The *retrieval threshold* parameter defines the activation level, below which a chunk can no longer be retrieved. By default the retrieval threshold is set to zero, thus the time of retrieval failure is e^{-S_p}.

If a production rule matches more than one chunk, then the latency is simply a sum of retrieval times of all the chunks in the production:

$$R_p = \sum_i t_{ip} \,.$$

2.3 Learning

If ACT–R could not learn, it would not be a good candidate for a theory of mind. Thanks to a number of learning mechanisms ACT–R can acquire information in the form of symbols and subsymbolic values and transform dramatically the knowledge of a model during its run. In this section only the learning mechanisms that are important for this work will be outlined. The descriptions of other mechanisms, such as association strength learning, which is not used here, can be found in the ACT–R book (Anderson & Lebiere, 1998, Chapter 4).

2.3.1 Learning Symbols

The initial contents of the declarative memory of an ACT–R model can be encoded by the programmer. This will allow ACT–R to start working on a problem. However, chunks may be added and modified by ACT–R's learning mechanisms during the model run. New chunks can be acquired from the outside world if ACT–R has ways of interacting with it, for example using the perceptual–motor extensions of ACT–R (Byrne & Anderson, 1998) or the Eye and Hand (Baxter & Ritter, 1996; Ritter et al., 2000) mechanisms of the Tower of Nottingham simulation (Jones et al., 2000). The model described in the next chapter uses

[1] Instead of chunk activation A_i, the retrieval time definition in the ACT–R book (Anderson & Lebiere, 1998) uses the match score M_{ip}, which is used for partial matching mechanism. In case of a perfect match $M_{ip} = A_i$. Because in this work partial matching is not used, we write A_i in the retrieval equation.

its own simple perception–action model capable of creating and modifying chunks in the model memory based on the information in the simulated environment.

The production rules should also be encoded beforehand by the programmer, but there is a mechanism in ACT–R, which allows it to learn new rules. This mechanism is called *production compilation*, which takes its roots from the analogy mechanism of the earlier versions of ACT–R, and has a long history. Production compilation is quite different from the chunking mechanism in SOAR(Rosenbloom & Newell, 1987). There are more constraints and consequently more limitations. The constraints, however, allow modellers more control over what is learned in the models.

ACT–R follows the proceduralisation theory of skill acquisition, that is procedural knowledge is formed from declarative. Particularly, from declarative representations of dependencies between events or goal states. These representations are encoded in ACT–R in the form of chunks of a special type *dependency*, which become the skeletons for new production rules. Production compilation is described in detail in the ACT–R book (Anderson & Lebiere, 1998).

2.3.2 Learning the Conflict Resolution Parameters

When a goal or a subgoal is completed, it is removed from the stack. A goal can be removed from the stack with two outcomes: success or failure.[2] A success occurs in the following situations:

1. When the goal is removed by a rule with the explicit `:success` flag.
2. When the goal is removed by any other rule without the `:failure` flag.

A failure is registered in one of the following situations:

1. When the goal is removed by a rule with the explicit `:failure` flag.
2. When the goal is removed because no productions have been found to match the current goal (instantiation failure).

ACT–R keeps track of which productions set the goals in the first place. So, when a goal is completed, ACT–R can update the number of successful or failed completions of

[2]It is also possibile to have a neutral outcome which does not change anything.

the goal for the corresponding rule. Each rule has its number of *successes* and *failures*. Using these numbers ACT–R can empirically estimate the probability P_i in equation (2.1):

$$P_i = \frac{\text{Successes}_i}{\text{Successes}_i + \text{Failures}_i}. \tag{2.5}$$

The number of successes by default is set to 1, thus the initial value of probability of any rule $P_i = 1$. Anderson and Lebiere (1998, page 135) explain that it makes prospects of new productions optimistic. This approach, however, is biased, and it will be discussed in Chapter 5.

ACT–R also keeps track of the time elapsed between the times at which a goal was pushed on and popped off the stack. Thus the average cost of the production can also be estimated:

$$C_i = \frac{\text{Efforts}_i}{\text{Successes}_i + \text{Failures}_i}, \tag{2.6}$$

where efforts is the cumulative time spent in all instances the production was used.

The described above mechanism of learning probabilities P_i and costs C_i in ACT–R is called *parameters learning*. The initial values of successes, failures, and efforts can be set directly, which is useful when the probabilities and costs should be set in advance, or change slowly.

It has been noticed in many animal learning experiments that not only the number of successes and failures is important, but also their timing (Baum, 1973; Mark & Gallistel, 1994). Particularly, more recent outcomes play a greater role in determining the choice behaviour. It has been proposed that animals are matching the reinforcement rate by estimating the rate of a Poisson process (Myerson & Miezin, 1980). ACT–R has a mechanism for *events discounting* making recent successes and failures more important than the older ones. As a result the probabilities and costs become decaying functions over time, and the learning is more adaptable (see Lovett & Anderson, 1996; Lovett, 1998). Due to the great computational overhead of the events discounting mechanism it was not used in this study. However, the new conflict resolution algorithm proposed in Chapter 6 uses posterior Poisson distribution to estimate the expected costs of production rules. Thus, it implements the idea more directly. Also, the algorithm is computationally cheaper.

2.3.3 Learning Chunk Activations and Production Strengths

If a chunk is not used, then its activation base–level decays according to equation (2.3). However, the base–level may increase each time the chunk is retrieved. This mechanism is

defined by the *base–level learning* equation:

$$B_i = \ln\left(\sum_j t_j^{-d}\right) + \beta \,, \tag{2.7}$$

where $t_1, t_2, \ldots, t_n$ are the time lags since chunk i has been used. As a result of this, learning chunks that are used more frequently have high activations, and activations of the rarely needed chunks decay below the retrieval threshold.

A similar mechanism is used for the production *strength learning*:

$$S_p = \ln\left(\sum_j t_j^{-d}\right) + \beta \,. \tag{2.8}$$

As was mentioned earlier, the decay rates d in equations (2.7) and (2.8) are set independently. The default values of d are .5.

2.4 Changes to the ACT–R Architecture

Some of the definitions described in this chapter do not match exactly the definitions in the ACT–R book (Anderson & Lebiere, 1998). Particularly, the expected probability was defined originally as a product of sub–probabilities q and r, where q was the probability that the rule fires if it is selected, and r was the probability that the goal would be achieved. The definition of $P = qr$, although seeming to be justified, led to a lot of problems in models, such as a decrease of P for rules, which should have been more successful. The q part of the expected probability has been abandoned in ACT–R Version 5.

Another change in ACT–R Version 5 is related to G parameter (goal value). In ACT–R 4 the mechanism called *goal discounting* was subtracting the cost of a rule from the goal value of a subgoal. This resulted sometimes in negative goal values in deep goal stacks. Because such behaviour led to undesired effects, the goal discounting mechanism was also abandoned in Version 5.

The problems described above were encountered by the author during the design of the model described in the next chapter. The model was written before the release of ACT–R 5. Two of the significant improvements included in ACT–R 5 were separately coded and included in the model described in the next chapter. Particularly, two hook–functions realise the changes described in this section (see Appendix B).

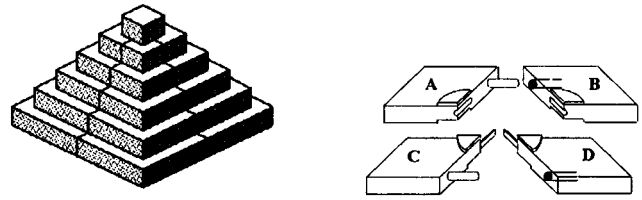

FIGURE 2.4: The Tower of Nottingham and four of 21 wooden blocks used to build the Tower of Nottingham (Wood & Middleton, 1975)

2.5 The Tower of Nottingham Model

Many ideas for this work were inspired by the results of the Tower of Nottingham model (Jones et al., 2000), which was used to study cognitive development. It is hard to create a model of a young learner. Children's behaviour is often very hard to predict. The idea used by Jones was first to create a fair model of adult subjects assembling the Tower of Nottingham puzzle (Wood & Middleton, 1975) shown on Figure 2.4, and then to achieve a match with the data from seven–year old children by modifying the original adult model.

The model was implemented in ACT–R Version 3, and it used the Nottingham Eye and Hand perception–action module (Baxter & Ritter, 1996; Ritter et al., 2000) to interact with the task simulation. The model achieved a fair match with the data from adult subjects at default parameters settings. Then, in order to match young problem solvers, Jones modified several architectural parameters, such as number of chunks in working memory, retrieval threshold, fovea and parafovea sizes (perception), and other parameters. All the modifications tried were based on prominent theories of development.

It is not necessary to describe here all the results of Jones' work. However, there was one particular adjustment that deserves our special attention. It was the model with increased noise ξ in the conflict resolution mechanism (see equation (2.1)). This simple modification alone produced excellent results. Other single manipulations could not produce such a good model fit. The EGN6 model (with parameter `:egn` set to 6.0) produced a particularly good match for the data, such as time needed to complete each layer and the number of constructions assembled on each layer (Figure 2.5 from Jones et al., 2000).

The fact that children seemed to be more 'noisy' or, speaking in ACT–R terms, less rational than adults, led to some interesting speculations and questions. For example, it may indicate that emotions play a greater role in children's learning and problem solving

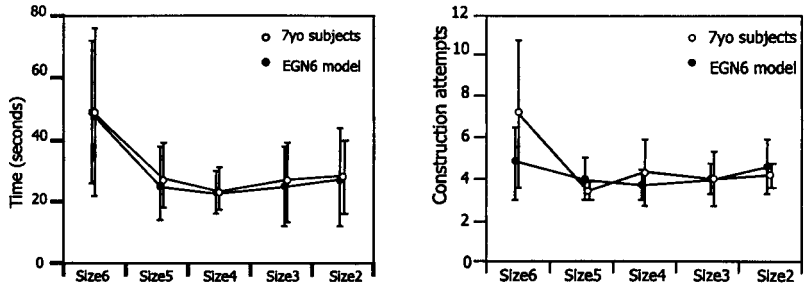

FIGURE 2.5: Time taken to assemble each layer and number of constructions assembled on each layer of the Tower by children and EGN6 model (from Jones et al., 2000).

(Belavkin et al., 1999). Indeed, joy and frustration are more observable in children. Another question was whether we could model even younger children by further increasing the noise. Or could these or better results be produced by modifying some other parameters in conflict resolution?

An attempt to model 3–4 year old children's behaviour by simply increasing the noise did not lead to the desired result: although the model as the 3–4 year old children could no longer solve the problem (Belavkin et al., 1999) it would not give up on a hopeless task. The model could ran for several simulated hours, which obviously is very far from the behaviour of 3–4 years old children who cannot solve the puzzle. Perhaps, part of the problem may have been addressed to the limitations of the model. However, the inability of ACT–R to evaluate its own performance in such situation may also indicate that there is a problem with the architecture.

Finally, the model showed some consistent discrepancies with the data: the number of errors and constructions for the first layer was usually a bit lower than for the subjects. This mismatch could be fixed by a further increase of the expected gain noise. However, this would result in a longer time needed by the model to solve the problem and many more errors during assembling the other layers of the Tower. It was suggested that noise in conflict resolution should decay towards the end of the task (Belavkin, 2001).

The results of the Tower of Nottingham pointed to the need for a closer study of the ACT–R conflict resolution mechanism and the influence of its parameters on problem solving.

2.6 Properties of the ACT–R Conflict Resolution

In this section the properties of the conflict resolution in ACT–R are analysed. The results of this analysis became the starting point in the research presented here. First, let us consider one example.

Example Imagine a simple and everyday situation: a pedestrian is walking along the main road, and suddenly his way is blocked by a car on a side–road, which has stopped to pass the vehicles on the main road. In this situation the pedestrian has at least two alternatives:

1. Wait until the car pulls out
2. Go around the car

We can model this situation in ACT–R, encoding the above alternative decisions in production rules. Because they both satisfy the same goal (to cross the side–road), we have a conflict, which in ACT–R will be resolved by comparing the expected gains of the rules.

Assume that executing the first rule, given that the car pulls out immediately, will take 5 seconds ($C_1 = 5$), and 10 seconds will be required to go around the car ($C_2 = 10$). This information may have been learned from previous experience. Because it is not certain that the car is going to pull out immediately, the expected probability of the first rule is less than for the second. Let $P_1 = .5$ and $P_2 = .8$. Then for $G = 20$ seconds (the default value) the mean values of the expected gains will be $E_1 = .5 \cdot 20 - 5 = 5$ and $E_2 = .8 \cdot 20 - 10 = 6$. So, for $G = 20$ the second rule will have higher priority because $E_1 < E_2$.

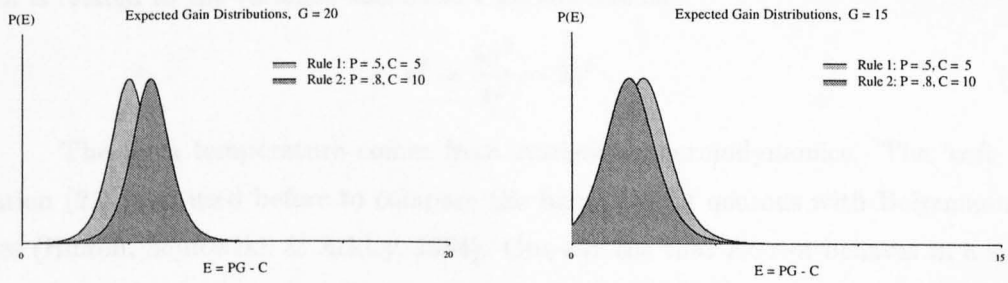

FIGURE 2.6: An example of how the priority of rules in conflict resolution changes for different values of G. Left: $G = 20$, $E_1 < E_2$; Right: $G = 15$, $E_1 > E_2$.

Not only expected probabilities and costs of rules affect the conflict resolution. In the above example one can easily check that for lower goal values, such as $G = 15$, the first rule will have higher expected gain, and it will be selected, despite its lower expected probability (Figure 2.6, right). Also, if the noise variance is high the choice will become more random and less dependent on Ps and Cs of rules in the conflict set.

The distributions of expected gains of the two rules with noise $s = .56$ are shown on Figure 2.6. The left plot illustrates distributions when $G = 20$, and for $G = 15$ on the right plot. One can see from Figure 2.6 that changing the goal value can be compared with changing the perspective at which we look at the two distributions, which can be compared with two hills. Such an analogy with a change of perspective was used by Tversky and Kahneman (1981) in their theory of decision framing, when they explained the difference in decision priority of subjects for the same problem presented in a different context.

2.6.1 Asymptotic Analysis of Choice Probability

In the example described above we have seen how the choice of a rule can be affected by the goal value and noise. Let us put these observations into a more precise mathematical form.

Consider a conflict set of n production rules. The probability $P(i)$ that rule i will be selected is given in closed–form approximation by the Boltzmann 'soft–max' equation:

$$P(i) = \frac{e^{E_i/\tau}}{\sum_j e^{E_j/\tau}}, \tag{2.9}$$

where E_i is the evaluation of ith rule $(P_i G - C_i)$, and τ is called the *noise temperature*, which is related to the variance and noise s parameters as:

$$\tau^2 = \frac{6\sigma^2}{\pi^2} = 2s^2 \ . \tag{2.10}$$

The term temperature comes from statistical thermodynamics. The 'soft–max' equation (2.9) was used before to compare the behaviour of neurons with Boltzmann machines (Hinton, Sejnowski, & Ackley, 1984). One can see that ACT–R behaves in a similar way, and τ in this context describes its temperature.

Let us analyse how the choice probability $P(i)$ depends on P_i and C_i for extremely high or low goal values G and noise temperature τ. Consider the simplest case of only two

rules ($n = 2$). Then the probability of choosing the first of the two rules will be:

$$P(1) = \frac{e^{(P_1G-C_1)/\tau}}{e^{(P_1G-C_1)/\tau} + e^{(P_2G-C_2)/\tau}}.$$

The behaviour of the system can be predicted by the asymptotic values of the above function at extreme values of G and τ:

1. Let $\tau \to 0$ (no noise), G = constant = 1. Then

$$P(1) \to 1 \quad \text{for} \quad E_1 > E_2$$
$$P(1) \to 0 \quad \text{for} \quad E_1 < E_2 \ .$$

   The behaviour in this case is completely deterministic. Note that if $P_1 = P_2$ and $C_1 = C_2$, then $E_1 = E_2$ and none of the rules will have an advantage (all rules initially have the same default values of probabilities and costs). We speculate that the system in this case behaves similarly to Elliot, a patient described by Damasio as unable to choose from equal opportunities and learning slowly (Damasio, 1994). It will be shown in Chapter 5 that noise not only helps to choose randomly, but also can accelerate learning.

2. Let $\tau \to \infty$ (high noise), G = constant = 1. Then

$$P(1) \to \frac{1}{2}, \quad \forall P_i, C_i$$

   In this case the choice becomes completely random. It does not depend on the past experience at all.

3. Let $G \to 0$ (low goal value), τ = constant = 1. Then

$$P(1) \to 1 \quad \text{for} \quad C_1 < C_2 \ , \quad \forall P_i$$
$$P(1) \to 0 \quad \text{for} \quad C_1 > C_2 \ , \quad \forall P_i \ .$$

   The choice is determined only by costs C_i and does not depend on probabilities P_i. Thus, a rule with a lower cost will always win regardless of its expected probability. The system behaves as if it is trying to put in as little effort as possible and does not 'care' about the success.

4. Let $G \to \infty$ (high goal value), τ = constant = 1. In this case

$$P(1) \to 1 \quad \text{for} \quad P_1 > P_2 \ , \quad \forall C_i$$
$$P(1) \to 0 \quad \text{for} \quad P_1 < P_2 \ , \quad \forall C_i \ .$$

This case is opposite to the previous one: the choice does not depend on costs C_i, and is determined only by the expected probabilities P_i. The system does not pay attention to the effort (time) it spends trying to achieve the goal whatever the cost.

Properties 1–4 are useful because they can predict the behaviour of models with different settings of the conflict resolution parameters. We shall use these properties throughout this study. For example, a model with dynamical G and τ will be presented in Chapter 5. The asymptotic properties will be used to analyse the advantages of the dynamical model.

2.6.2 Noise Relative to the Goal Value

In the *Performance* chapter of the ACT–R book (Anderson et al., 1998), while discussing the probability matching experiments, the authors point to the importance of the G/τ ratio. Indeed, property 4 suggests that when noise is high, it is possible to match the probability better by increasing goal value G. In the Tower of Nottingham model (Jones et al., 2000) the goal value was set to the default value $G = 20$ in all experiments. One may wonder whether the results of the EGN6 model, shown on Figure 2.5, could be reproduced by decreasing G instead of increasing the noise. Let us consider what happens to the choice probability (2.9) if we increase or decrease G and τ simultaneously, while keeping the ratio G/τ constant.

Let $\tau \to \infty$ and $G \to \infty$. In this case, like in 3, asymptotes of $P(1)$ are determined by the value of P_1:

$$P(1) \to \frac{e}{e+1} \quad \text{for} \quad P_1 = 1$$
$$P(1) \to \frac{1}{e+1} \quad \text{for} \quad P_1 = 0 \ .$$

So, if we increase both G and τ keeping the ratio G/τ constant, then probabilities P_i become more important than costs C_i ($P(1) > P(2)$ if $P_1 > P_2$ because $e \approx 2.71 > 1$).

Similarly, for both $G \to 0$ and $\tau \to 0$ the costs become more important. It means that even when G/τ is constant the value of G changes the choice strategy emphasising the

influence of either the probabilities (high G) or the costs (low G). Now, returning to the Tower of Nottingham question, one may see that decrease of G with the same proportion of noise as in EGN6 model would more likely produce different results.

It is convenient to normalise the noise temperature with respect to G. Let us define the *relative noise* as:

$$T = \frac{1}{G}\tau \cdot 100\% \ . \tag{2.11}$$

One may see that relative noise fixates the G/τ ratio. For example, $T = 20\%$ gives $G/\tau = 1/5$ for any G.

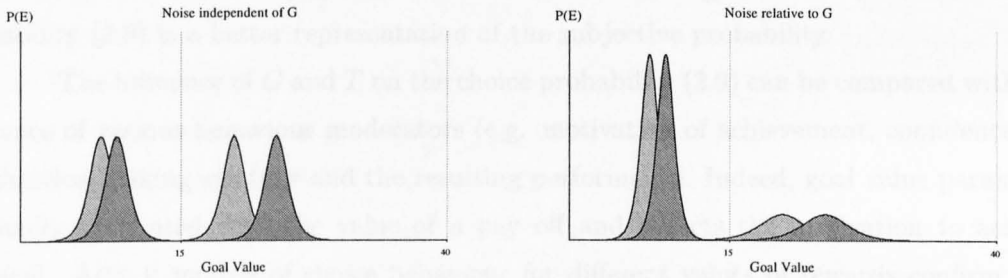

FIGURE 2.7: Distributions of expected gains as a function of G. Left: noise is independent of G; Right: noise is relative to G. On each plot two close distributions are shown for $G = 15$ and $G = 40$.

Figure 2.7 shows close distributions of expected gains for different values of G when noise is independent of G (left), and when noise is relative to G (right). One may see that with independent noise, the increase of G from 15 to 40 separates the distributions further from each other. When noise is relative to G, the increase of G separates the means, but also changes the variances proportionally.

In this work the terms relative noise T and noise temperature τ will be used to describe the noise settings. The variance and s parameter values can be found using equations (2.11) and (2.10).

2.7 Decision Making and the Principle Components of Emotions

The performance of subjects in a certain task is determined in the end by which decisions or actions they take. It is well–known that human decision making is based more on some simple heuristics rather than on sophisticated statistical analysis of probability distributions (Tversky & Kahneman, 1974). The so–called *subjective probability* is greatly determined by

the context in which a problem is presented. This phenomenon has been described as the *framing* of decisions by Tversky and Kahneman (1981).

As was mentioned earlier and shown in the example, the choice in ACT–R does not rely entirely on the experience stored in the form of Ps and Cs of the rules. Asymptotic properties 1—4 show precisely how the choice can be influenced by two other parameters: goal value (G) and noise (T). Thus, the subjective probabilities influencing subjects' choice should not be confused with the empirical probabilities P_i in the expected gain equation (2.1). Rather the expected gains and their distribution can be viewed as an ACT–R representation of one's subjective state of mind determining the choice, and the choice probability (2.9) is a better representation of the subjective probability.

The influence of G and T on the choice probability (2.9) can be compared with the influence of various behaviour moderators (e.g. motivation of achievement, confidence) on the decision making strategy and the resulting performance. Indeed, goal value parameter is usually associated with the value of a pay–off and relfects the motivation to achieve the goal. ACT–R models of choice behaviour for different values of rewards confirm that idea (see Figure 2.3). Moreover, the aversive stimulation will result in a goal to avoid the stimulus, and high G can also represent this situation for a high motivation to avoid the loss.

Different levels of stimulation (physical, mental, chemical, etc) are known to be related to the level of arousal. As mentioned earlier, this multidimensional measure is used to reflect the different levels of activation of the autonomic nervous system, which can be influenced by many factors (Thayer, 1978; Humphreys & Revelle, 1984). Different levels of arousal are resulting in different types of behaviour: low cost, energy saving actions at a low arousal, while more energy intense actions characterise a high arousal. Properties 3 and 4 show that ACT–R avoids costly decisions under low G and is more likely to spend more efforts for high G. This suggests that the behaviour in a low or high arousal states can be modelled in ACT–R using low and high goal values.[3]

Confidence, or the 'feeling of knowing', is important for the success in problem solving. Properties 1 and 2 show that at high noise the choice is less dependent on the empirically learned information. Perhaps the amount of relative noise could be associated with the confidence of a problem solver. Indeed, two models with exactly the same knowledge

[3]Note that high noise in the conflict resolution can also result in choosing costly productions. However, use of relative noise T makes this effect negligible.

and probability settings would behave differently under different noise settings. A good example is the Tower of Nottingham model when noise increase in the same model leads to more errors.

Furthermore, Tversky and Kahneman (1981) point out that behaviour of subjects in a choice involving a loss is risk taking (more random). In ACT–R this corresponds to the conflict resolution with high noise. On the contrary, if a choice involves gains, then the pattern of subjects' behaviour is risk aversive (less random). This situation can be modelled in ACT–R with low noise settings. Losses and gains are accompanied by negative and positive emotional experiences. Thus, the conflict resolution in ACT–R with high or low relative noise T can represent the decision making of subjects influenced by the negative or positive emotions respectively.

Despite the great variety of emotions, most of their classifications follow the two–dimensional pattern (e.g. Russell, 1983, 1989). The first dimension is the strength or intensity of the emotion, and it is associated with the arousal. The second is called the *valence* dividing the emotions into negative or positive. Throughout this thesis we shall refer to arousal and valence as the *principle components emotions.* Hopefully, this distinction (principle components of emotions rather than emotions) will help us to reduce the ambiguity of the mapping from emotions to the parameters of ACT–R. The summary of relations between different behaviour moderators discussed in this section and the values of ACT–R conflict resolution parameters is suggested in Table 2.1.

TABLE 2.1: Conflict resolution settings and types of choice behaviour.

G/T settings	Corresponding behaviour
low T	risk aversive, choice involving gains, high confidence, positive valence of emotions
high T	risk taking, choice involving losses, low confidence, negative valence of emotions
low G	spending less efforts, low motivation, low level of stimulation, low arousal
high G	ready to spend more efforts, high motivation, high level of stimulation, high arousal

2.8 ACT–R and the Inverted–U Effect

The inverted–U hypothesis relating arousal to performance has been a subject of many debates. Humphreys and Revelle (1984) suggested that the maximum of cognitive performance for a particular level of stimulation can be explained by the multidimensional nature of arousal. They argued that if arousal affects several cognitive processes, such as 'information transfer' and 'memory'[4], then the effects of these processes on performance may be different for particular levels of arousal and may even have the opposing dependencies. The result of the combination of several factors is an inverted–U function with a maximum in one point.

The performance of an ACT–R cognitive model can be affected by many parameters, such as activations of chunks, strengths of productions, and conflict resolution parameters. If these parameters depend differently on the levels of stimulation, then the model will express the inverted–U behaviour. For example, Figure 2.8 illustrates a model of the inverted–U law explained by the dynamics of G and τ in ACT–R: an increase of stimulation (arousal) corresponds to an increase of both G and τ; the growth rate of G and τ, however, is not the same, and the performance is better (e.g. the fastest learning), when the ratio G/τ has its maximum value.

Using the data of the most famous experiment on the inverted–U effect — the Yerkes and Dodson experiment — we shall study the above hypothesis by modifying several architectural parameters of ACT–R.

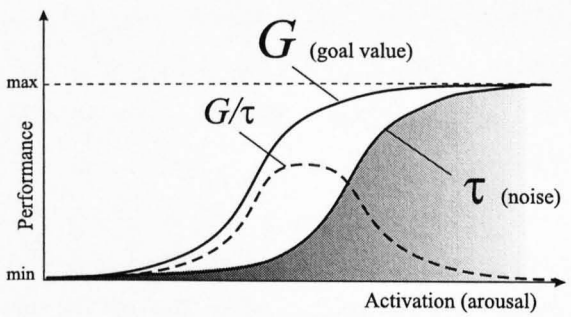

FIGURE 2.8: A model of inverted–U effect using ACT–R conflict resolution parameters G (goal value) and τ (noise temperature). The best performance corresponds to the maximum of G/τ ratio.

[4]It is not quite clear, however, to which parameters in ACT–R the terms 'information transfer' and 'memory' used by Humphreys and Revelle correspond.

2.9 Summary

In this chapter the main features and mechanisms of the ACT–R cognitive architecture have been described. Some important results of several cognitive models have been discussed in order to identify the areas in ACT–R theory of particular interest for this research. Specifically, the conflict resolution mechanism has been analysed and compared with some other decision making theories, and a way to model the effects of arousal and valence on decision making has been proposed. The most important variables that can significantly influence the performance of an ACT–R model and contribute to the inverted–U effect have been described. Table 2.2 provides a summary of these variables, their ACT–R parameters names, and related equations.

TABLE 2.2: Summary of important variables and equations

Var.	Name	Equations	Parameter	Default val.
P_i	expected probability	(2.1), (2.5)		1.0
C_i	average cost	(2.1), (2.6)		.05 sec
G	goal value	(2.1)	`:g`	20 sec
s	expected gain noise s	(2.1)	`:egs`	`nil`
τ	noise temperature	(2.9), (2.10)		
T	relative noise	(2.11)		
β	base level constant	(2.3)	`:blc`	0
s_2	chunk activation noise s	(2.3)	`:ans`	`nil`
d	base–level decay rate	(2.7)	`:bll`	.5
d	production strength decay rate	(2.8)	`:sl`	.5

CHAPTER 3

A Model of the Yerkes and Dodson Experiment

In this chapter a model of the Yerkes and Dodson experiment will be described. In this experiment, as has been described earlier, they studied learning in mice in a two–choice task. Yerkes and Dodson called their experiment a 'dancing mouse' task, and referred to mice as 'dancers'. For this reason the model was also named 'Dancer'.

It is necessary to clarify that this is not the first time when ACT–R, officially a theory of human mind, is used to model the behaviour of animals. For example, Lovett and Anderson (1995, 1996) used ACT–R to model choice behaviour of pigeons (Herrnstein, 1961). Moreover, the main properties of choice behaviour of animals and human subjects (i.e. probability matching, dependency on reinforcement) are quite similar (see Lovett, 1998). Because the Yerkes and Dodson experiment is a simple two–choice task, modelling it with ACT–R seems plausible.

The Dancer model consists of two main parts: an ACT–R cognitive model of the mouse and task simulation with its graphical representation and user interface. The first section of this chapter will be dedicated to the design concept of the cognitive model, its organisation, components, and operation. The perception, decision making, and action stages of the model will be outlined. The outline of experiments and calibration of model parameters will be explained in the second section. The goals, strategies, learning and the resulting behaviour of the mouse will be explained in the third section. In addition, the complete code of the model is presented and described in Appendix A.

3.1 Model Overview

Any model is a simplification of the reality. The assumptions made in the beginning of designing a model are crucial for both the success of the experiment and the correct interpretation of its results. In this section the main objectives that motivated the design and assumptions of the model will be outlined. The level of details in which the mouse and the environment is represented is motivated not only by the requirements of this study, but also by the possibility of future reuse of the model.

3.1.1 Objectives

The main goal of building the model is not to explain the data of the Yerkes and Dodson experiment as best as possible, but rather to see how the inverted–U effect can be obtained using different parameters of the ACT–R architecture.

The task of the Yerkes and Dodson experiment is relatively simple: the mouse has to learn to always choose the door with the white card on it regardless of whether the door is on the left or right. The data collected in the experiments is the number of errors produced by the mouse. The main result of the Yerkes and Dodson work is that moderate levels of stimulation, and hence arousal, result in better performance than at low or more extreme values. The inverted–U effect was studied and observed by psychologists in many other studies using similar experiments (e.g. Mandler & Sarason, 1952; Näätänen, 1973; Gupta, 1977; Anderson & Revelle, 1982; Matthews, 1985).

In the previous chapter using asymptotic analysis we predicted the effect of different values of parameters in the conflict resolution on performance of ACT–R models. We are going to build a model, where these predictions can be tested experimentally. In order to do this we need a task where a goal can be achieved using different strategies with different costs and success probabilities. The choice of these strategies depends on which production rules are selected during the conflict resolution. Because the choice of strategy affects the performance of the model, we are going to study how the performance depends on G and T parameters, and if it does, then what the dependency looks like, and where is the point of optimal performance. In addition, the effect of other mechanisms of ACT–R, such as the production strength learning, will be investigated.

Finally, the way the inverted–U effect occurs should be reusable in other tasks and experiments. In order to achieve that the knowledge representation model, goal generation

scheme, and production rules must be as abstract as possible, and not related to the specific task. It needs to be fundamentally in the architecture. To make the model look more realistic the behaviour of the mouse should be emergent based on the mouse motivations and conditions of the environment.

3.1.2 Model Assumptions

Because our interest is mainly in the area of learning and performance, the model has rather simple assumptions about perception and action. In particular, we assume that the mouse can always distinguish between the black and the white cards on the doors regardless of the lighting conditions, and the effect of these conditions is in the way this information is used for learning.

Indeed, lateral inhibition in the perceptual system of animals makes the edges and differences between the brightness of objects more pronounced (Hartline, Wagner, & Ratcliff, 1956). So, we assume that although mice do not have particularly good sight in general, their visual system must be sufficient to see where the darker door is located. However, we assume that the darker conditions should make it less obvious to use the colour information when learning new rules.

3.1.3 Task Simulation

The task simulation was developed for the model using Common Lisp and the GARNET GUI library (Myers et al., 1990). The user interface consists of several windows. The main window shown on Figure 3.1 represents the discrimination chamber with the Dancer (a mouse in the Yerkes and Dodson experiment) and three rooms (the main room and two escape boxes). The doors of the escape boxes are marked by white and black cards. The location of white and black cards can be random, or controlled by the predefined order such as in the original experiment (see Table 3.1).

Other windows in the simulation include the control panel window with gadgets to set up the parameters of the model, and several windows for real–time graphical representation of the results.

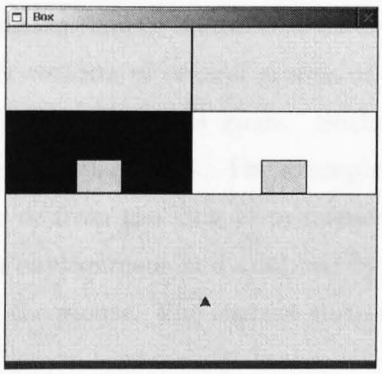

FIGURE 3.1: The main window representing the discrimination chamber with two escape doors.

3.1.4 Model Organisation

The model can be divided into three modules based on their functionality: perception, cognition, and action. These modules, their components, and information flows between them are shown schematically on Figure 3.2.

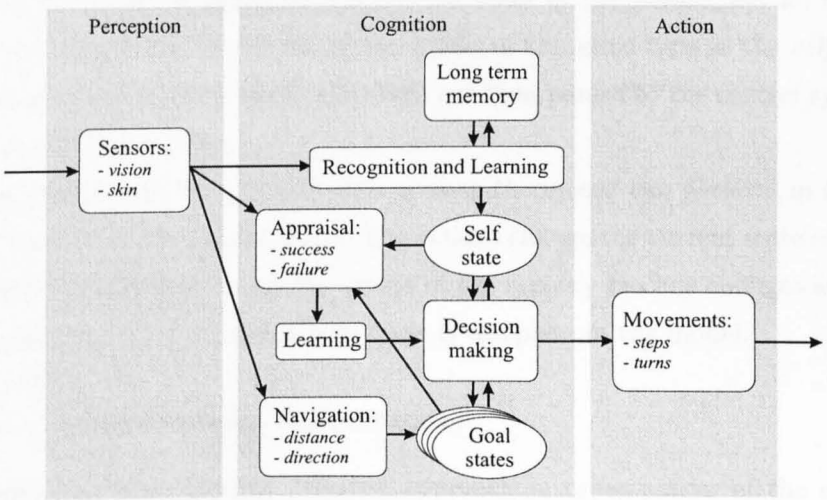

FIGURE 3.2: Block diagram of the main modules and information flows in the Dancer model.

Information about the environment is gathered in the perception module, which includes a rather simple set of sensors: visual and skin sensors. Also a particular set of rules in the model is responsible for determining the distance and direction to an object in the environment when the mouse is attending it. This is the navigation subsystem. Both

sensory and navigation information require interaction with the simulated environment.

The cognitive module consists of several groups of production rules with specific functionality, and acting on specific types of goals. Such a distinction allows dividing the cognitive module into several subsystems. For example, the recognition system may receive a goal from perception or from the long term memory (declarative memory). The information gathered from the environment and analysed by recognition system is used to determine the current state of the mouse. The current state of the mouse is represented by one chunk `self0`.

Several production rules are responsible for assessing the success or failure of a previously set goal based on the current state. These rules represent the appraisal system. The successes or failures information is used to learn new production parameters (see equations (2.5) and (2.6)).

Some production rules may add new declarative memory elements (chunks) to the long term memory, while another set of rules use ACT–R's production compilation mechanism to learn new production rules. These rules represent the learning subsystem.

New goals are set based on the current state of the mouse and its motivations. Goal states are represented in the model by chunks of the same type as the *self* chunk, but with the desired values in their slots. The goals are then passed to the control system which generates commands for action.

The last module is responsible for actions the mouse can perform in the environment. These actions are steps and turns. The actions change the current state of the mouse, and as a result the information on the inputs of the sensory module changes as well. This way the perception, cognition and action cycle is complete in the model.

3.1.5 Object Oriented Knowledge Representation

ACT–R enables the use of object oriented approach in organisation of the chunk–types (classes of declarative knowledge units). By means of the `:include` command chunk–types may inherit properties of their parent, more general chunk–types. This allows creating a simple, and still quite powerful hierarchical knowledge representation of objects in the simulated environment. In addition, it allows the generalisation of the production rules, because in ACT–R rules operate not only on goals of a specified chunk–type, but also on all its subclasses.

For example, the most elementary object in the model is a point in space and time represented by a chunk–type *location*. All other objects are represented in the model by chunks of types that inherit the *location* chunk–type. Although different classes of objects in the simulated world, such as rooms and doors, are represented by chunks of different types, the production system uses one single rule called *object–location* to determine the distance and direction to all objects, because it looks for a goal of the *location* type.

3.1.6 Blackboards Concept and Means-Ends Analysis

The most important declarative knowledge elements in the model are chunks representing states of the mouse itself. These are chunks of the type *self*, which is a subclass of the *object* type (see Appendix A for definitions). There is always at least one chunk of this type in the working memory: chunk `self0` representing the current state of the mouse. The values of the slots of this chunk represent different relations of the mouse with the outside world, such as which objects are around, which box it is currently in, and whether the mouse feels good or bad, and so on. Many production rules in the model analyse the information from the perceptual inputs to fill in the specified slots of the current state chunk. Although there are many rules in the system operating on chunks of type *self*, the production system is organised in such a way that usually the rules look only into specific slots of these chunks. Such organisation of production systems is known as blackboards concept (Erman, Hayes-Roth, Lesser, & Reddy, 1980), and chunks of type *self* here serve as the blackboard.

The information gathered from the environment and analysed is used to generate the goal states. If something in the current state does not correspond to the desired state, then the mouse tries to minimise this difference. This principle is known as means-ends analysis, and it is used throughout in the model to make the behaviour of the mouse emergent from the difference between the current and the desired states.

3.1.7 Perception, Cognition and Action Loop

The model uses serial organisation where each of the three modules works one after another in sequence: the perceptual, cognitive, and, finally, action cycles. Figure 3.3 shows schematically this process.

In order to separate production rules of the perception, cognition, and action

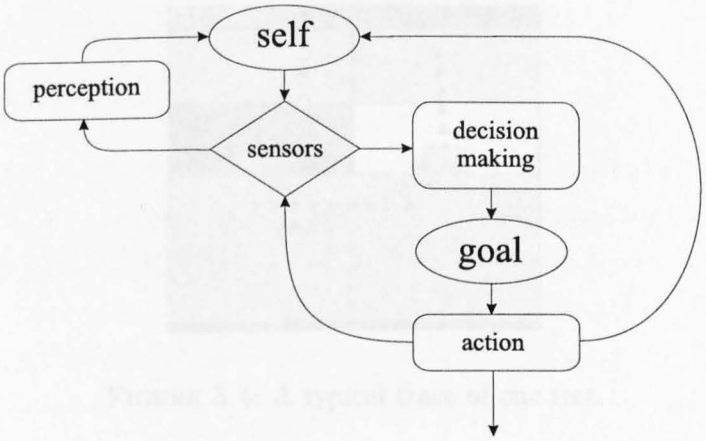

FIGURE 3.3: Block diagram presenting three stages of the model: perception, cognition (decision making), and action.

modules from each other, the rules act on goals of different types. Rules from the perceptual module act on goals of type *perception*, while rules from the action module need goals of type *action* (see Appendix A for the definition). Goals used by rules of the cognitive module are mainly representations of objects such as current state of the mouse, goal states, rooms, exits, etc.

The model uses chunk *sensors* (the only chunk of type perception holding the current outputs of all the sensors in the perceptual system) as a predicate to determine whether it should enter the perceptual or cognitive cycle. Starting from chunks of type *self* representing the current or goal states of the mouse, the model checks for the values of the slots in the *sensors* chunk: if the values are `nil`, then the *sensors* chunk becomes the focus of the model and the model automatically enters the perceptual cycle because only the rules from the perceptual module operate this type of goal.

After the information from the environment has been gathered, the focus returns to the goal of type *self*. Now, using the new information from the perceptual system, rules from the cognitive module may alter the representation of the mouse own current state, representation of some objects in the environment, assess the completion of previously set goals, learn new rules, and possibly create new goal states. The difference between the current and the goal state is analysed and the goals of type *action* are created.

Finally, the model enters the action cycle because rules from the action module operate on goals of type *action*. Rules from the action module make calls to the simulation

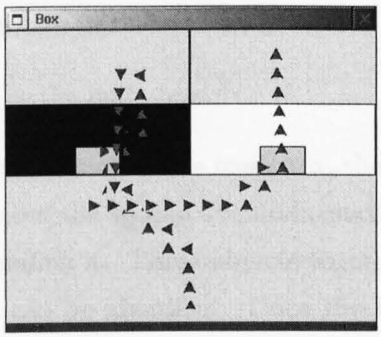

FIGURE 3.4: A typical trace of one test.

which perform the required state changes, and also set the values of the sensors to `nil`. Thus, the model automatically proceeds to the perception cycle again after the action has been completed.

The serial organisation of the model excludes interaction between perception, cognition and action separating them into three independent stages. This seriality is artificial, but is sufficient and useful simplification for the purposes of this study. In future the model can be extended taking advantages of the ACT–R 5 and its perceptual–motor extensions.

3.2 Running Experiments Using the Model

In this section we outline what the model is doing in each test, what the experiments consist of and what we measure in these experiments. In addition, the problem of calibration model parameters will be discussed.

3.2.1 Outline of One Test

In each test of the Yerkes and Dodson experiment the mouse was required to escape the discrimination chamber through one of the escape doors. Figure 3.4 illustrates a trace of one model run simulating this test.

On the screen the Dancer is represented by an arrow object. Starting in the centre of the main box the dancer analyses information obtained from the visual system about what it sees ahead, and from the skin sensors about the level of external stimulus. The analysed information is gathered in slots of the chunk `self0` representing the current state of the dancer. The current state is then compared against the two objectives:

1. The level of external stimulus must be zero.

2. The mouse must not be in the main box.

Because there is no stimulation in the main box, the first objective is fulfilled from the beginning. In order to meet the second requirement the mouse has to look around and see all the objects surrounding it. These objects together form a chunk representing the environment, which then can be identified. Once the box the mouse is in has been recognised as the main box, the model creates a new goal to escape the box. This goal is represented by a chunk of the same type as the current state chunk. The difference is that the goal state describes the mouse placed in the complement of the current environment (not in the main box).

When the model has a goal to escape the current room, the dancer looks for the exits. There are two exits from the main box, and at this point the model has to make the crucial decision: which exit to choose. Figure 3.4 shows the case when the dancer chose the left door with a black card on it. Once the dancer enters the left box the skin sensors immediately detect the high level of external stimulus (an electrical shock in the original experiment). The visual and recognition systems register that the environment has changed and the mouse is no longer in the main box.

Although the second objective (to escape the main box) is complete, the high level of external stimulation signals a failure to achieve the top goal. At this point the model can learn new production rules to choose a different exit based on the information about the most recent choice it has made.

When the dancer detects some non–zero level of external stimulus it sets a new goal, which is a chunk describing a state with zero level. To achieve this goal the dancer can use two strategies: to move into another point in the same box, or to escape the current box altogether. Figure 3.4 shows the dancer first trying to move around several points inside the same box, and only after several failures escapes the box. The model learns that the first strategy, although is cheaper, does not lead to a success. The second strategy requires more efforts, but always achieves the goal.

When the mouse escapes the left box it automatically enters the main box back again. After the environment has been recognised the model creates a new goal to escape the main box again through one of its exits. The situation, however, is now different from the one in the beginning of the task: the model has learned after the last failure a new

production rule to choose another door. If this new rule fires, then the dancer will make the right choice. Because both new and old production rules compete in the conflict set, and there is some noise in the conflict resolution, there is a possibility that the dancer will choose the wrong door again. Figure 3.4 shows the dancer choosing on the second attempt the door on the right with the white card on it. Once the dancer enters the right box the system detects that both objectives are fulfilled and the task is completed.

3.2.2 Running Series of Tests

In their experiment Yerkes and Dodson subjected each mouse to 10 tests per day for up to 30 days (training series). In each test the position of the black door was changed according to Table 3.1. On the first two days (preference series) denoted by letters A and B there was no electrical shock behind the black door, and the mouse was allowed to escape the main box through any door. After the first two days (preference series) the mouse was only allowed to escape through the door with the white card, and if it attempted to escape through another door (with the black card) the aversive stimulus (electrical shock) was used. The number of errors for each day was recorded, and if the mouse did not produce any errors for three consecutive days, the experiment was terminated. The authors referred to this moment as the *perfect habit* formation.

As in the original experiment the model was run for up to 320 tests (2 training and 30 learning days with 10 tests per one simulated day) with the positions of the doors defined in Table 3.1. Figure 3.5 shows an example of a typical error curve obtained in one experiment (left) and a curve of simulated time required by each test (right). In this example the perfect habit was formed on simulated day 5. The average amount of time needed to complete each test in this example is 14.6 simulated seconds.

3.2.3 Calibration of Model Initial Parameters

Because the ACT–R learning mechanisms use time decaying variables (production strength, chunk activations, etc), setting realistic time parameters is very important for the final model performance.

The model needs on average 244 cycles to complete each test. Nevertheless, the amount of simulated time the model spends during these 244 cycles depends very much on retrieval times (2.4), which contribute to latencies between the cycles. The minimal

TABLE 3.1: Positions of white cardboards for two preference series (day A and B without stimulation) and twenty–five training series. Each series consists of 10 tests per day. Letter 'r' or 'l' denotes a position on the right or left (from Yerkes & Dodson, 1908).

Tests→ / Series↓	1	2	3	4	5	6	7	8	9	10
A	l	r	l	r	l	r	l	r	l	r
B	r	l	r	l	r	l	r	l	r	l
1	r	l	r	l	r	l	r	l	r	l
2	l	l	r	r	l	r	l	l	r	r
3	r	r	l	r	l	l	r	l	r	l
4	l	l	l	r	r	r	l	r	r	l
5	r	l	r	l	r	l	r	l	r	l
6	l	l	r	l	r	r	l	r	l	r
7	r	l	l	l	r	r	r	l	r	l
8	r	r	l	l	r	l	r	l	r	l
9	r	r	r	l	l	l	r	l	r	l
10	l	l	l	l	r	r	r	r	l	r
11	r	l	r	r	r	l	l	l	r	l
12	r	l	r	l	r	r	l	l	r	l
13	r	l	r	l	l	l	r	r	r	l
14	l	l	l	l	r	r	r	r	l	r
15	r	l	r	r	r	l	l	l	r	l
16	l	r	l	l	l	r	r	r	l	r
17	r	r	r	r	l	l	l	l	r	l
18	l	r	l	r	r	l	l	r	l	r
19	r	l	r	l	r	l	r	l	r	l
20	l	l	l	r	l	r	l	r	r	r
21	r	l	l	r	r	l	l	r	r	l
22	l	l	r	r	l	l	r	r	l	r
23	r	l	l	l	l	r	r	r	r	l
24	l	r	l	l	l	r	r	r	l	r
25	r	r	r	r	l	l	l	l	r	l

(default) action time in ACT–R is 50 milliseconds. So, if all the production rules fire at the minimal 50 milliseconds, then the model will spend on average 12 simulated seconds in each test.

As defined by the retrieval latency equation (2.4), the reaction times in ACT–R directly depend on two parameters: production strength S_p and chunk activation A_i. Low production strength values and low activations result in higher latencies, thus increasing the amount of simulated time needed by the model to complete each test. The production strength learning and the base level learning mechanisms of ACT–R implement the power

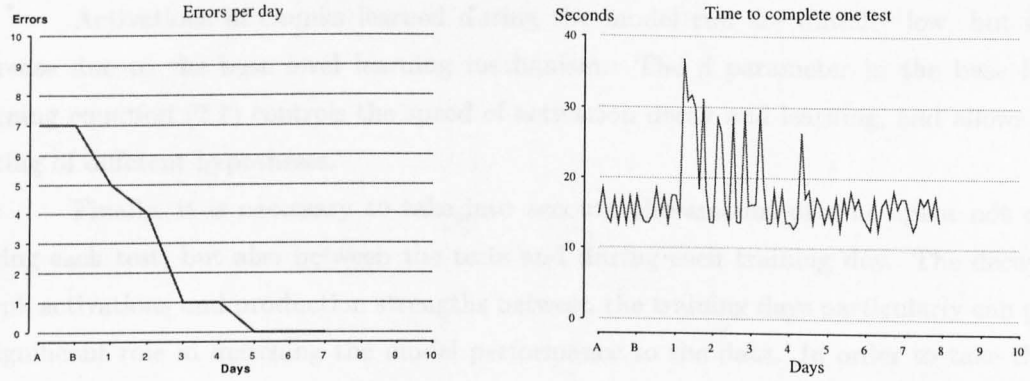

FIGURE 3.5: Typical error curve (left) and time trace obtained in one experiment. In this example the model needed 50 tests to form the perfect habit (no errors produced after testing day 5).

law of practice increasing strengths of rules and activations of chunks used more frequently. This way with practice the reaction times in the model decrease.

The set of production rules the model starts with simulates a very basic knowledge that the mouse must have used during all its life. Indeed, these are productions responsible for orientation, navigation, exploration, basic movements control, etc, so the latencies of these rules should be minimal. It is achieved by setting high values of production strengths directly or, if production strength learning mechanism is switched on, by setting the *creation times* and *references* parameters of the productions (see Appendix A).

New production rules learned by the model initially have low strength, and hence higher latencies. However, with practice the strengths may increase due to the strength learning mechanism. Using parameter d in the strength learning equation (2.8) controls the speed of strength decay and learning, which allows the testing of different hypotheses. The effect of this parameter on model performance will be explained in the next chapter together with the results of the S_p–model.

Many production rules use two chunks in the model: chunk `self0` describing the current state of the mouse, and chunk *sensors* representing the sensory inputs. Because these chunks are used practically in every production rule, their activations must be very high, and they should not affect the reaction times. Activations can be set directly or, if the base level learning mechanism is switched on, by setting the *creation time* and *references* parameters of the chunks (see Appendix A).

Activations of chunks learned during the model run are initially low, but may increase due to the base level learning mechanism. The d parameter in the base level learning equation (2.7) controls the speed of activation decay and learning, and allows the testing of different hypotheses.

Finally, it is necessary to take into account the amount of time spent not only during each test, but also between the tests and during each training day. The decay of chunk activations and production strengths between the training days particularly can play a significant role in matching the model performance to the data. In order to take these effects into account the ACT–R simulated time in the model is increased by 60 seconds between each test, and by a large time interval (an approximation of several hours) between each simulated day.

3.2.4 Mapping the Conflict Resolution Parameters

The level of the electric stimulus in the original experiment was varied from 125 to 500 in Martin's units of stimulation (Yerkes & Dodson, 1908). The authors distinguished between three levels of stimulation: weak ($50 \sim 150$), medium ($150 \sim 250$), and strong ($250 \sim 500$). There is currently no parameter in ACT–R that can represent the level of external stimulation, but because this external stimulation forces the mouse to set a goal to escape, we may roughly relate different levels of stimulation to different values of this goal (parameter G in ACT–R). The control panel has a gauge gadget to set the goal value (parameter G) of ACT–R. Obviously there is no conversion table from Martin's units to values of G parameter, so it was necessary to find the most appropriate values empirically.

Each model run (one test) requires on average 244 cycles to complete, and ACT–R simulated time needed is approximately 15 simulated seconds. The costs of production rules usually do not exceed 30 seconds. Thus, the lowest goal value can be set to 20 seconds (the default value in ACT–R).

The next step is to determine the high boundary of goal values corresponding to the high levels of stimulation. One may suggest that the highest goal value is life, which is probably true considering that the high stimulation was electrical shock. Another approach to determine the highest values of G can take advantage of the asymptotic properties. Indeed, at high values of G the costs of production rules should have little influence on conflict resolution. The expected probabilities P, even if they are very small, must determine

the choice. Using the empirical information about the costs of production rules in the model one may set $G \approx 500 - 1000$ as the highest values. Indeed, suppose two rules compete in the conflict set, and let $C_1 = 30$ (high cost), $C_2 = 0$ (low cost). Let also the expected probabilities of both rules be very small, for example $P_1 = .2$ and $P_2 = .1$. Note that the probability of the first rule (with the highest cost) is slightly larger. Now, if $G = 500$ it is elementary to check that the expected gain $(PG - C)$ of the first rule will be greater:

$$.2 \cdot 500 - 30 = 70 > .1 \cdot 500 - 0 = 50 \, .$$

So, the goal value varies linearly from 20 (weak stimulation) to 500 (strong stimulation) seconds.

The model controls the expected gain noise s parameter of ACT–R through relative noise T equation (2.11). Thus, the increase of G increases noise temperature τ (and noise s) proportionally keeping the percentage of the noise in conflict resolution constant. Relative noise was varied in the model between $T = 1\%$ and 20%.

3.3 Learning in the Model

The model can use many of the learning mechanisms offered by the ACT–R architecture. Table 3.2 shows a summary of the learning processes in the model. The mechanisms can be first divided into two classes: symbolic and subsymbolic. While symbolic learning is concerned with creating new or modifying the existing knowledge units in the memory, the subsymbolic mechanisms are modifying and learning new values of various subsymbolic parameters of these knowledge units. Both symbolic and subsymbolic learning can be classified further according to whether it is related to procedural or declarative knowledge units (production rules and chunks).

TABLE 3.2: Summary of ACT–R learning mechanisms used in the model.

	Declarative	Procedural
Symbolic	Adding new chunks Modifying existing chunks	Learning new production rules
Subsymbolic	Base level learning	Probability learning Cost learning Production strength learning

3.3.1 Learning Declarative Memory Units

The model has the capacity to learn new chunks representing the objects in the outside world based on their visual properties. The recognition mechanism can distinguish between the new objects and those the model has seen before. However, sometimes the model may fail to recognise some objects (i.e. due to a failure to retrieve a chunk, or because of the noise in the conflict resolution). Thus, the model may create several chunks to denote the same object. Although this is not a problem for one test, in a longer run, such as 300 tests, the model can learn up to several thousand chunks. Due to the complexity in the working memory such learning creates it becomes extremely hard to debug the model and separate the effects produced on the model performance by parameters manipulation from the erratic behaviour of the overloaded program. For this reason the rule acquiring the representations of new objects was removed from the model during the tests, and when the model started it already had chunks representing the objects in the task (walls, doors and rooms).

The model, however, still needs the recognition mechanism, because it is important for learning the chunks' activations. The new chunks the model creates are mainly the new goal states. Once a goal is completed it can be removed from the memory or the chunk may merge with another chunk if it has identical type and slot values.

The values of chunk slots can be modified by production rules during the model run. For example slots of the current state chunks are rapidly updated as the mouse acquires new information about its own state. New values may determine what production rules the model will consider next.

3.3.2 Learning Declarative Memory Parameters

In order to recognise an object the mouse is observing ahead, it has to retrieve a chunk from memory describing an object with similar visual properties. The retrieval time depends on the chunk's activation as described by equation (2.4). The model can use the base level learning mechanism of ACT–R to learn new activation values, so the more often the chunk is retrieved the higher its activation becomes, and as a result the object is recognised faster. Figure 3.6 illustrates the dynamics of activations of all chunks in declarative memory of the model during 100 tests (the base level learning and activation noise were switched on).

The curves in the top part of Figure 3.6 represent high and slowly changing activations. These are of the chunks representing the current state of the mouse, the recognition

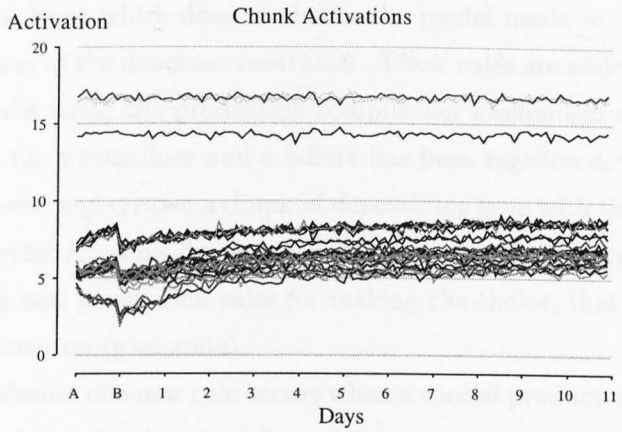

FIGURE 3.6: Dynamics of activations of all chunks in the model during 100 tests.

buffer, and the sensory input. The curves in the lower part of the graph are activations of chunks representing the objects in the environment and their properties.

The retrieval threshold, activation base level constant, and activation noise parameters are used in the model to create the errors of omission, that is when the mouse cannot retrieve certain chunks (e.g. chunks encoding the colour information). The d parameter in base level learning equation (2.7) can be used to control the speed of activation decay.

3.3.3 Production Compilation

When the mouse sets a goal to escape the main box, it faces a dilemma: which door to choose? A choice of one of two objects is represented in the model by a chunk of a special type *choice* (see Appendix A). The model starts with two simple production rules matching the choice chunk goal:

Choose1st:
IF the goal is a *choice* between *first* and *second*
THEN focus on *first*

Choose2nd:
IF the goal is a *choice* between *first* and *second*
THEN focus on *second*

When the choice chunk appears in the model as a goal, then one of the production rules above may fire selecting one of the alternatives. Thanks to the noise in the conflict resolution the two rules above realise random choice of the doors.

In order to learn which door to choose the model needs to learn new production rules that use features of the doors as constraints. These rules are added into the procedural memory of the model using the production compilation mechanism of ACT–R. After the dancer has entered the wrong door and a failure has been registered, the model recalls the last choice it has made, and creates a chunk of *dependency* type with the choice chunk in the *goal* slot (see Appendix A). This dependency chunk is used by the production compilation mechanism to learn new production rules for making the choice, that is to choose another door in the same situation (goal state).

The compilation of a new rule occurs when a special production rule (learning rule) considers the dependency chunk as a goal, modifies it, and pops it off the goal stack. There are three special learning rules in the model that can perform this task (see Appendix A for the precise code). These rules analyse the choice chunk, which contains the information about which door the dancer chose on the last attempt. A new production rule to be created should be similar to the two simple choice rules shown above: the new rule also matches the choice goal. The new rule, however, should also consider the properties of the objects in the choice as constraints, and based on these properties the rule should choose the object opposite to the one selected in the last attempt (because it led to the failure). Below is an example of a rule that the model may learn:

IF the goal is a *choice* between *first* and *second*
AND *first* is a *black* door
THEN focus on *second*

Note that the above rule uses only one feature of the objects — colour. Another feature is the door's position (left or right), and the model can learn to choose based on the position information:

IF the goal is a *choice* between *first* and *second*
AND *first* is a door on the *left*
THEN focus on *second*

Finally, the model may learn a rule that pays attention to both features:

IF the goal is a *choice* between *first* and *second*
AND *first* is a *black* door on the *left*
THEN focus on *second*

What kind of a new rule is learned depends on which of the three learning rules is used when the dependency chunk is in focus, and it is determined by the conflict resolution.

Two of the three learning rules create new productions paying attention only to one feature: colours or positions of the doors, such as the first two rules shown above. These two learning rules implement one–dimensional reflection learning. The third learning rule implements two–dimensional learning, that is the new production it creates uses both features as constraints: colours and positions of the doors, such as in the third rule shown above. A model of one and two dimensional learning using production compilation mechanism in ACT–R was first realised by Taatgen (1997).

Learning new production rules allows the model to make the choice considering more conditions. One may notice, however, that some of the rules the model learns are potentially more successful than the others. The first rule that pays attention only to the colour will be the most successful, because it realises the correct strategy: avoid the black door. The second rule has a 50% chance to succeed because the positions of the doors and are, in fact, random. The third rule will be successful every time it fires, but because the left–hand–side includes one unnecessary constraint (door position), there are fewer situations when it can be applied. The model learns which rules are more successful using the probability learning mechanism described below.

3.3.4 Learning Production Probabilities and Costs

The model learns both expected probabilities and costs of production rules. These parameters are used to calculate the expected gains of the rules using the utility equation (2.1) of the ACT–R conflict resolution. Figure 3.7 shows traces of probabilities P, costs C, and expected gain values ($PG - C$ for default $G = 20$) learned during the first 120 tests (12 days).

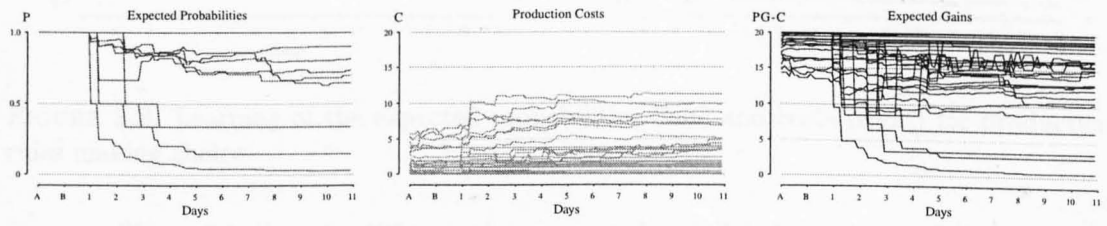

FIGURE 3.7: From left to right traces of expected probabilities, production costs, and expected gains as they are learned in the model during 120 tests (2 training and 10 testing days). Note a sudden change in values when the stimulation is switched on after the two training days.

Figure 3.7 shows values of P, C, and $PG - C$ of all the production rules in the model, but not all rules are competing on a given cycle. Thanks to the modular organisation of the model and the blackboards concept in each particular situation (particular goal state) there are usually only few conflicting rules. There are two conflicts in the model that are particularly important for the model performance:

1. A conflict between rules that make choice of one of the two doors.
2. A conflict between rules selecting the strategy to escape the stimulation in a room with the black card.

The first conflict occurs when the choice of one of the two doors appears as the goal. As was mentioned earlier, there are initially only two rules making this choice, but more rules can be learned later through production compilation. After one of these rules has fired, the dancer escapes the main box through the chosen door. The appraisal system of the model registers a success or a failure of the decision made previously. Using the parameters learning mechanism ACT–R learns empirically probabilities P and costs C of the production rules (see equations (2.5) and (2.6)).

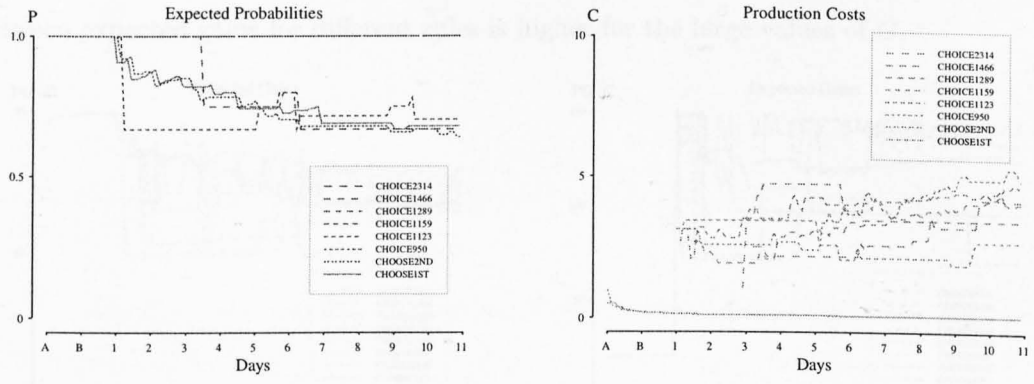

FIGURE 3.8: Learning of the expected probabilities (left) and costs (right) for production rules making choice.

Figure 3.8 shows for 120 tests the learning of probabilities and costs for productions making the choice. Note that during the two training days A and B (first 20 tests) there are only two rules participating in the conflict: *choose1st* and *choose2nd*. The probabilities and costs of these rules remain approximately the same for the first 20 tests. After the training the stimulation is switched on, the number of rules increases up to eight, and the number

of curves on the plots increases correspondingly. The new production rules are named by ACT–R automatically based on the goal type, which is *choice* in this case. Thus, new rules, which are learned by ACT–R spontaneously, have names such as CHOICE2314. The plots were generated automatically during the model run.

As was mentioned earlier the new rules learned by the model have more constraints. One can see from Figure 3.8 that costs of the new productions are higher than that of the two simple choice rules (*choose1st* and *choose2nd*). If the new rules, however, realise the successful strategy (i.e. the strategy to avoid a door with the black card), then their success probability will be high ($P = 1$). The model may also learn a wrong strategy. One can see from the figure that probabilities of some new rules quickly decay and reach values close to $P = .5$. Probability learning mechanism allows the model to distinguish between rules that implement more or less successful strategy.

The performance of the model will depend on how well it can distinguish between high and low probabilities. Asymptotic properties of conflict resolution in ACT–R imply that higher goal values should improve the distinction by probabilities even if the relative noise remains the same. Figure 3.9 shows examples of learning expected gains for $G = 20$ (left) and $G = 150$ (right), and the relative noise $T = 10\%$. One can see that variance between expected gains for different rules is higher for the large values of G.

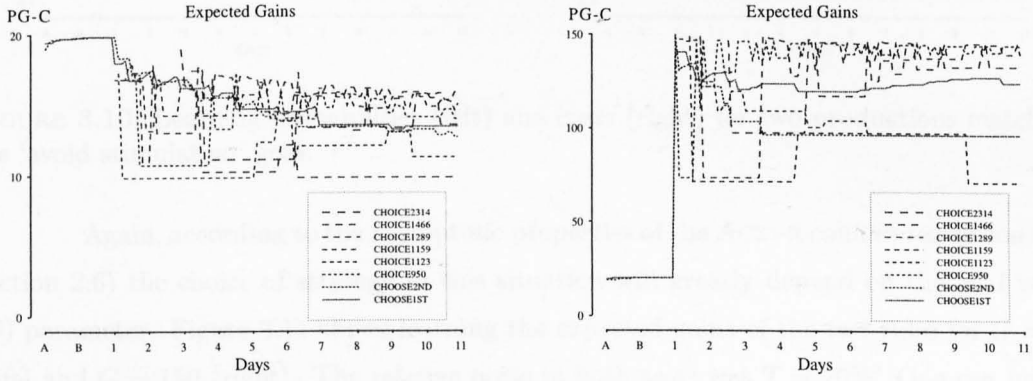

FIGURE 3.9: Learning the expected gains for $G = 20$ (left) and $G = 150$ (right). The relative noise $T = 10\%$ in both examples. Note how the difference between high and low expected gains increases with G.

The second conflict occurs when the mouse enters the box with a simulated electrical shock. The model then immediately sets a goal to escape the stimulation. There are

two production rules in the model that match this goal state. These rules differ only in the right–hand sides realising different strategies:

Find Better Point:
IF the goal is a state with no stimulation
THEN set a new goal to move into another point within the same box;

Find Better Environment:
IF the goal is a state with no stimulation
THEN set a new goal to escape the box altogether.

Unlike the latter rule, the first rule sets the goal which requires only few steps to complete. Hence the model learns that the effort required by the first strategy is always less than the cost of escaping the box. However, in this task the first rule can never achieve success, and the model learns that the expected probability of the first rule is much smaller than that of the second rule. Figure 3.10 shows learning probabilities and costs of the above two rules.

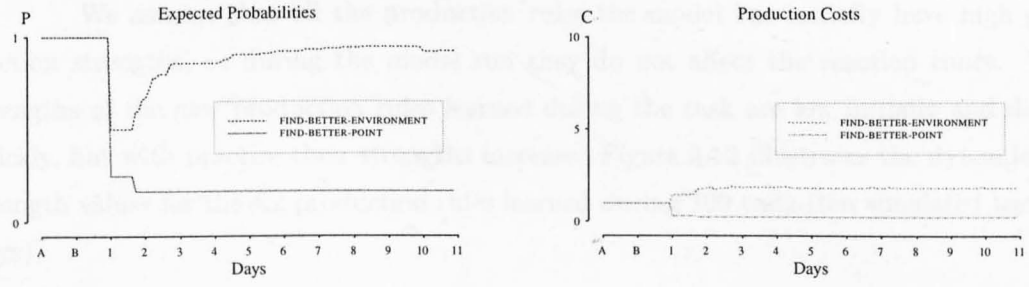

FIGURE 3.10: Learning probabilities (left) and costs (right) for two productions matching the 'avoid stimulation' goal.

Again, according to the asymptotic properties of the ACT–R conflict resolution (see Section 2.6) the choice of strategy in this situation will greatly depend on the goal value (G) parameter. Figure 3.11 shows learning the expected gains of the two rules for $G = 20$ (left) and $G = 150$ (right). The relative noise in both tests was $T = 20\%$. One can notice that at low goal values the difference between expected gains of both rules is marginal, and although the *find better point* rule may never lead to the success, the model still may often try this rule. The resulting behaviour is that after entering the wrong box at low G values the dancer does not immediately escape the box, and wanders around attempting to find a better point inside it. On the contrary, at high G the dancer immediately escapes the electrical box.

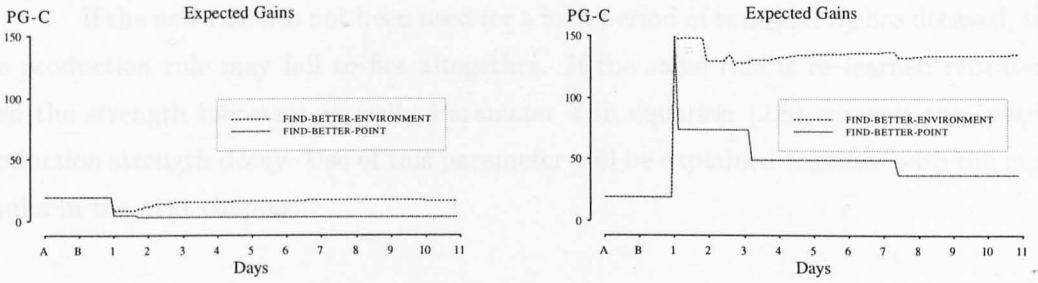

FIGURE 3.11: Learning the expected gains for two productions matching the 'avoid stimulation' goal with $G = 20$ (left) and $G = 150$ (right).

3.3.5 Learning Production Strength

The model can use the production strength (S_p) learning mechanism (see equation (2.8)). Production strength (S_p) of a rule determines how fast it fires, and affects the reaction time as explained by the ACT–R retrieval latency equation (2.4).

We assume that all the production rules the model has initially have high production strengths, so during the model run they do not affect the reaction times. The strengths of the new production rules learned during the task are low initially and decay quickly, but with practice their strengths increase. Figure 3.12 illustrates the dynamics of strength values for the six production rules learned during 100 tests (ten simulated testing days).

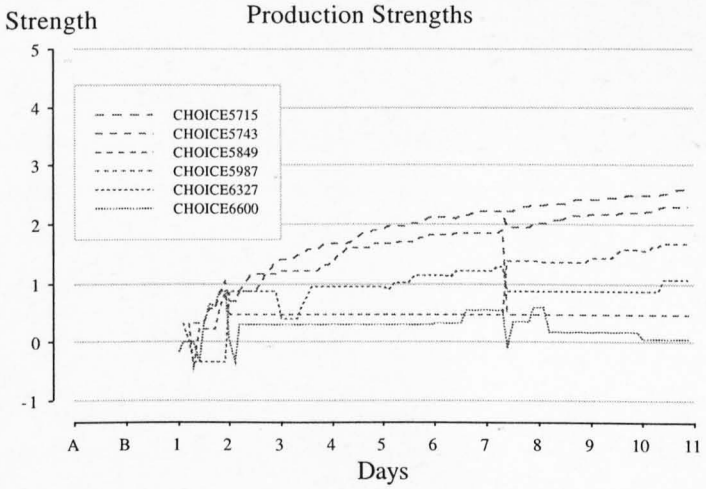

FIGURE 3.12: Dynamics of strengths (S_p) of new productions learned during 100 tests.

If the new rule has not been used for a long period of time and S_p has decayed, then the production rule may fail to fire altogether. If the same rule is re–learned repeatedly, then the strength increases as well. Parameter d in equation (2.8) controls the speed of production strength decay. Use of this parameter will be explained together with the model results in the next chapter.

3.4 Summary

We described the main features of the Yerkes and Dodson experiment simulation and the Dancer model. The general behaviour of the model during each test as well as the learning processes have been outlined. In the next chapter the results of series of tests ran at different parameters settings will be presented. The results will be compared against the data from the Yerkes and Dodson experiment. The data consists of three sets: set I (medium visual discrimination), set II (good visual discrimination) and set III (slight visual discrimination). Analysis of the comparison of the model results with each test will be presented. Finally, the U–shaped performance curves obtained by Yerkes and Dodson in set I and set III of the experiments will be compared with the matching models.

CHAPTER 4

Model Results and Analysis

In this chapter the results of the Dancer model will be presented and analysed. The performance of the model under different settings will be compared with the data from Yerkes and Dodson (1908). It is important to note, however, that achieving the perfect fit with the data was never the goal of this work. In this chapter the model behaviour will be studied as a function of several subsymbolic parameters in the ACT–R architecture. In particular, four crucial parameters will be manipulated: expected gain noise variance, goal value, production strength learning and base–level activation of colour chunks. Investigating just five values of each parameter results in $5^4 = 625$ combinations, which may take quite a long time to investigate: each experiment consisting of up to 300 tests takes about one minute to complete on 700MHz processor computer; for each parameters setup the model runs at least 20 experiments; thus, in ideal situation of continuous run it takes $\approx$ 208 hours of machine time (more than eight days) to complete all the tests. For this reason the parameters were manipulated with quite a rough precision (e.g. relative noise values of T = 1%, 5%, 10% and 20%). The data was used for reference points indicating how the model fit changes with regards to parameters manipulation. So, while the errors reported for the best models will seem to be quite high (e.g. 10%), it is important to remember that these are not the smallest errors possible.

4.1 The Data

The original paper reports results (records of errors produced by mice in the experiments) of three experimental sets with different conditions of visual discrimination: medium (set I), good (set II), and slight discrimination (set III). In each set the performance of mice was

measured for several levels of electrical stimulation. Table 4.1 shows the levels of stimulation and the corresponding number of subjects used for each of the three experimental sets. The errors for each subject were reported, and then the average number of errors produced by the mice in a particular experimental setup (condition and strength of stimulus) were shown on charts. These experimental distributions of errors will be shown in this chapter on charts along with the errors produced by the corresponding models (see Figures 4.6, 4.14 and 4.15).

TABLE 4.1: Levels of stimulation and number of subjects used for each condition of discrimination and training (from Yerkes & Dodson, 1908).

Condition of discrimination	Strength of stimulus	N. of subjects
Set I (Medium, Medium)	125	4
	300	4
	500	4
Set II (Good, Easy)	135	4
	195	4
	255	4
	375	4
	420	4
Set III (Slight, Difficult)	135	2
	195	2
	255	2
	375	2

Figure 4.1 (based on data from Yerkes & Dodson, 1908) shows the main results of the experiments: the numbers of tests required to form the perfect habit (no errors for three consecutive days). Abscissae represent levels of stimulation, and ordinates represent the average number of tests required. Three curves corresponds to data of set I, II, and III (medium, good and slight discrimination). One can see that in set I and III the best performance was at moderate levels of stimulation. In set II the performance improved almost linearly the level of stimulation. Thus, the inverted–U effect was only observed when visual discrimination was not perfect.

First, the model will be compared with data from set II (good visual discrimination). We start with set II because it has the most precise data (five different levels of stimulation as opposed to three and four in set I and III). Furthermore, the data of set II is easier to model because visual discrimination is high, and the relation between the level of stimulation and performance is simpler (linear) than in set I or III.

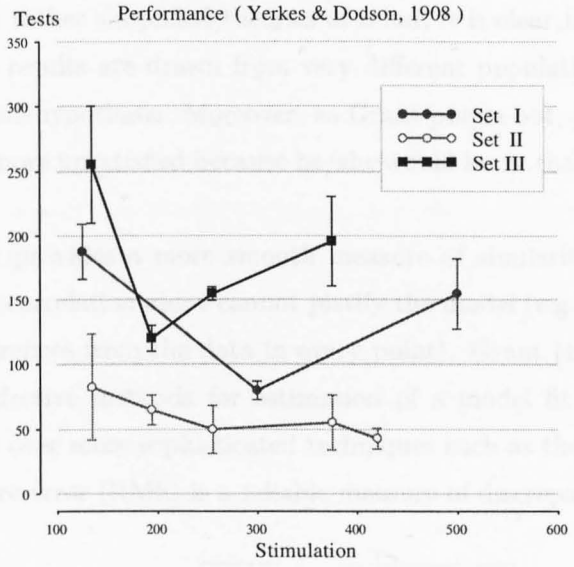

FIGURE 4.1: Performance curves (from Yerkes & Dodson, 1908): ordinates represent the average number of tests required to form the perfect habit; abscissae represent levels of stimulation. Three curves represent results of set I, II, and III (medium, good and slight discrimination).

The first model will study only the influence of the ACT–R conflict resolution parameters on task performance. It will be shown that increase of stimulation corresponds to an increase of goal value G and possibly also to an increase of relative noise T. The second model will consider the implications of different production strength learning rates on performance. Finally, a way to simulate different conditions of visual discrimination by modifying activations of colour chunks in the working memory will be introduced. The third model will be compared with the results of set I and set III. It will be shown how the model can reproduce the inverted–U effect observed in these experiments.

4.2 Criteria for Comparing Model Results with the Data

Fitting cognitive models to a data collected in a small–scale, insensitive experiment is a task where standard statistical tests, such as the null–hypothesis (H_0) test, may lead to bizarre conclusions (Grant, 1962). Although the kind of test we are looking for is similarity between theoretical model and data, the attempt to prove H_0 can be self–defeating. Indeed, because the data is very noisy and the models (as well the architectures they are implemented in)

are usually based on rather simplified theories of mind, it is clear from the beginning that the data and model results are drawn from very different populations. Thus, there is no need to reject the null–hypothesis. Moreover, as Grant points out, proving H_0 would leave the researcher even more unsatisfied because he/she would know that the result is not really reflecting the truth.

Correlation provides a more smooth measure of similarity of a model with the data. However, high correlation alone cannot justify the model (e.g. the model results may have consistent differences from the data in every point). Grant (1962) points out several simple, but quite effective methods for estimation of a model fit. These methods have practical advantages over more sophisticated techniques such as the H_0 test. For example, the root–mean–square error (RMS) is a reliable measure of discrepancy between the model and the data:

$$\mathrm{RMS} = \sqrt{E\{e_i^2\}} = \sqrt{E\{(Y_i' - Y_i)^2\}} \ ,$$

where Y_i' are theoretical and Y_i are data points. Another excellent index for evaluation of the model performance the variance of errors:

$$\sigma_D^2 = E\{(e_i - E\{e_i\})^2\} \ .$$

Another method suggested by Grant is estimating the discrepancy at individual points. This is particularly valuable in our case because the variance of the data is not homogeneous: the deviation of the errors in the final stage of the experiment is smaller than at the beginning. Thus a model with significant discrepancy in the final stage of the run (usually after 10th simulated day) will be particularly suspect, and a model with a smaller correlation and a slightly higher RMS may be preferable if the discrepancy in the final stage is smaller.

In this chapter the root–mean–square errors (RMS) will be reported as percentage of the maximum error a model can produce (i.e. maximum error of 10 corresponds to RMS = 100%). In our investigation a model that produced a smaller error will be preferable over a model with a higher correlation. The Dancer model managed to produce candidates with RMS of 5% — 10%.

Although an error of 10% may seem generous, it is important to remember that these errors were achieved with quite a rough parameters manipulation procedure. In addition, as can be seen from Table 4.1, the data itself is based on a very limited number of measurements, which is quite common for psychological experiments.

Finally, linear regression of the model on data will be calculated and plotted on a regression graph. An example is shown on Figure 4.2. The coefficient of determination R^2 will be reported as it is usually a better estimation of model performance than the correlation coefficient. Figure 4.2 is produced by fitting a data set to itself, and it is an example of a perfect model: the points are perfectly situated on the regression line and the plot is symmetrical. Ideally RMS = 0% and $R^2 = 1$. The mean of the data points is shown by the vertical line, and the mean of the theoretical points by the horizontal line. Their intersection, the centre of gravity, is another important criterion: ideally the model should produce on average the same number of errors as the subjects.

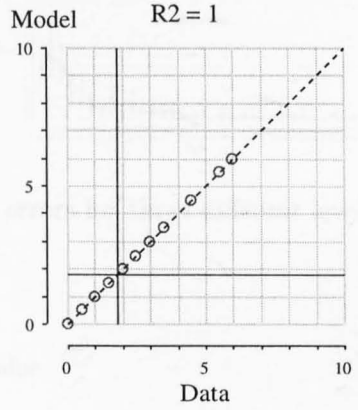

FIGURE 4.2: Linear regression for a theoretical perfect model: $R^2 = 1$, RMS = 0%.

4.3 The G/T–Model

The model described in this section was used to study the effect of conflict resolution parameters, namely the goal value G and relative noise T, on the model performance. On the subsymbolic level only the expected probabilities and costs of the production rules were learned. The production strength and the base level learning mechanisms were switched off and did not influence the results. Because the model does not take into account the possibility of low visual discrimination, the results will be first compared with the data for good visual discrimination (set II, Yerkes & Dodson, 1908).

4.3.1 Influence of the Expected Gain Noise

The conflict resolution mechanism of ACT–R selects the rule with the highest expected gain (or utility) value determined by the equation (2.1). However, when the noise variance is extremely high, the rules with low expected probabilities and high costs may still win over the more successful and efficient rules, thus leading to performance degradation. Figure 4.3 illustrates distributions of errors (based on 20 model runs) produced by the model during 320 tests (two training and 30 testing simulated days) for three levels of relative noise in conflict resolution: $T = 20\%$, $T = 10\%$, and $T = 1\%$. One may see that as the noise decreases the overall performance of the model improves, which corresponds to the predictions.

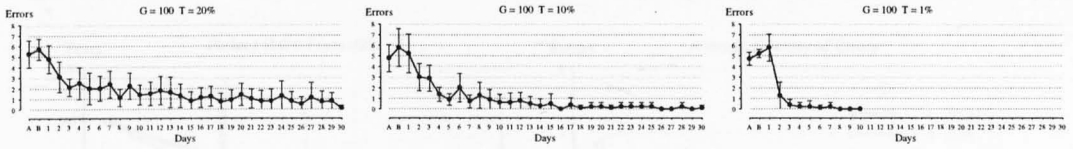

FIGURE 4.3: Distributions of errors for three different levels of relative noise. From left to right $T = 20\%$, 10%, and 1%.

4.3.2 Influence of the Goal Value

The goal value (G) in the utility equation (2.1) determines the maximum mean value of expected gain E. It was shown earlier in the asymptotic analysis of conflict resolution

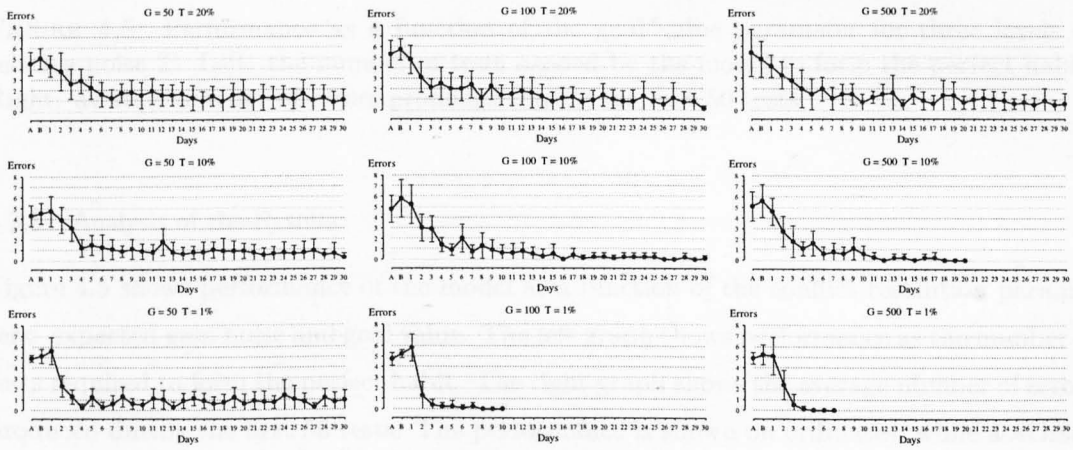

FIGURE 4.4: Distributions of errors for three goal values (from left to right $G = 50$, 100, and 500), and for three levels of relative noise (from bottom to top $T = 1\%$, 10%, and 20%).

(see Section 2.6) that high values of G emphasise the influence of P, while low G values make C more important. Moreover, this effect is observable even if the relative noise $T = \frac{1}{G}\tau$ remains constant. Thus, the decrease of the goal value should reduce the effect of probabilities on the outcomes of conflict resolution. This should lead to degradation of performance, because the success of the model in each test depends on selection of the rules with high probabilities.

Figure 4.4 demonstrates the distributions of errors for three goal values (from left to right $G = 50$, 100, and $G = 500$), and for three levels of relative noise (from top to bottom $T = 20\%$, 10%, and 1%). As expected the performance of the model improves with the goal value increase. Note that the effect is particularly noticeable for low noise.

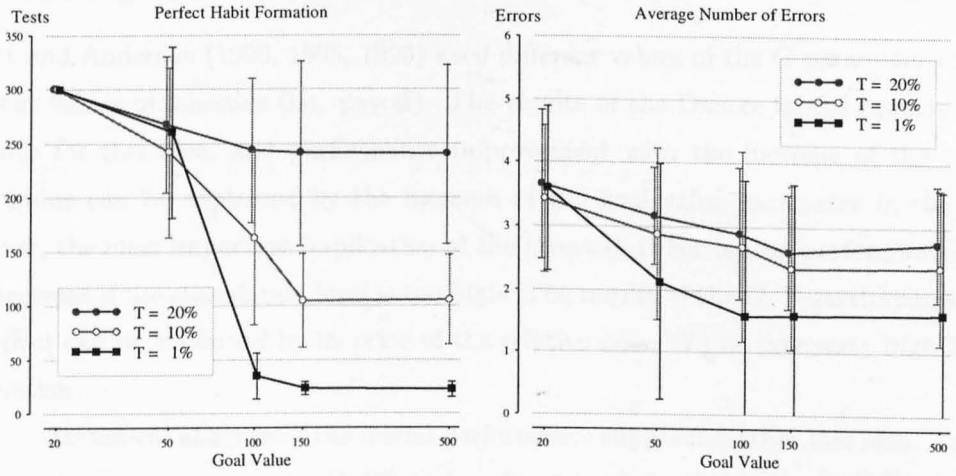

FIGURE 4.5: Performance as a function of the goal value parameter for three levels of relative noise T. Left: the number of tests needed by the model to form the perfect habit. Right: average number of errors produced during the first 50 tests.

4.3.3 Analysis of the Results

Figure 4.5 shows performance of the model as a function of the conflict resolution parameters: expected gain noise and goal value. The left graph shows performance as the number of tests required to form the perfect habit. The right graph shows the average number of errors produced during the first 50 tests. The performance is shown on ordinates, while abscissae represent different goal values. Different curves correspond to different noise settings. One can see that increase of the goal value dramatically improves the performance, while higher

noise leads to performance degradation. The effect of the noise is more pronounced at the high goal values, while the effect of the goal value is more noticeable for lower noise.

Note that noise increase leads to a greater variance on the left plot (number of tests needed to form the perfect habit), but to a smaller variance on the right plot (the number of errors during the first 50 tests). When model results are compared with the data a model producing on average the same performance usually has much greater variances. One might suggest that in order to achieve the same variation as in the data the model would probably need to start with a greater noise, but should have less noise closer to the end of the experiment.

4.3.4 Matching the Data of Set II

Lovett and Anderson (1990, 1995, 1996) used different values of the G parameter to model different values of stimulus (i.e. payoff). The results of the Dancer model provide further evidence for this idea, and performance improvement with the increase of the strength of stimulus can be explained by the increase of the goal value parameter in the model. However, the most important implication of the inverted–U law is that performance may in fact decrease if the stimulation level is too high. The results of the G/T–model suggest that this effect can be explained by increase of the relative noise (T) at extremely high levels of stimulation.

Statistical analysis of the model performance supports further this idea. Table 4.2 shows root mean square errors (RMS) and coefficients of determination (R^2) for the G/T–model ran at different G and T settings compared with the Yerkes and Dodson data from set II. Good matches in terms described in Section 4.2 (minimising RMS, maximising R^2, etc) are shown in bold. One can see from the table that increase of stimulation from 135 to 420 units corresponds to increase of the goal value (in our case from 50 to 500). Figure 4.6 shows the learning curves for the selected in Table 4.2 model settings compared with the data of set II (left) and corresponding regression plots (right).

Note that in the data of set II there was no noticeable degradation of performance at high levels of stimulation. One can see from Table 4.2 that all the best matches are for the noise setting $T = 10\%$. Nevertheless, at high levels of stimulation (375 and 420) the model with higher noise setting $T = 20\%$ and $G = 500$ provides better correlation than the model with lower noise. Thus, relative noise is more likely to increase at extremely

high levels of stimulation. Later in this chapter a model taking into account the different conditions of discrimination will be introduced. The results of this model compared with the data from set I and III further support the idea that noise increases at a high level of stimulation, which also explains the degradation of performance (i.e. the inverted–U effect).

The increase of noise can be understood taking into account the circumstances of the Yerkes and Dodson experiment. The stimulation was an electrical shock, which at high values should have resulted in a lot of fear of an error in mice because the penalty was so severe. It was noted earlier that high relative noise in conflict resolution may correspond to the behaviour with low confidence (see Table 2.1), which explains why the model suggests the noise increase at high levels of stimulation.

TABLE 4.2: Comparison of the G/T–model with the data from set II (good visual discrimination). Data for different stimulation levels compared with model performance under various levels of noise (T) and goal values (G). The best matches in terms described in Section 4.2 are shown in bold and on Figure 4.6. Increase of stimulation corresponds to increase of G and possibly T at high levels of stimulation.

Stimulation		135		195		255		375		420	
T	G	rms	R^2	rms	R^2	rms	R^2	rms	R^2	rms	R^2
	20	19.7%	.67	18.5%	.73	19.8%	.66	19.5%	.58	17.3%	.76
	50	9.6%	.58	9.6%	.72	10.0%	.78	13.0%	.53	10.0%	.67
1%	100	14.3%	.58	11.3%	.80	12.8%	.76	16.9%	.50	12.5%	.67
	150	17.8%	.59	15.2%	.78	14.6%	.90	21.0%	.51	13.9%	.70
	500	16.1%	.62	14.2%	.77	13.0%	.92	18.9%	.55	12.3%	.71
	20	9.9%	.70	11.4%	.89	14.5%	.70	15.0%	.58	11.9%	.69
	50	11.8%	.55	12.9%	.78	15.8%	.61	16.9%	.44	14.0%	.57
5%	100	11.8%	.54	13.6%	.77	15.4%	.72	17.1%	.46	14.3%	.62
	150	13.1%	.45	15.8%	.66	17.9%	.58	18.7%	.34	16.4%	.50
	500	13.6%	.53	15.7%	.74	17.8%	.66	18.3%	.44	16.1%	.57
	20	15.4%	.70	14.9%	.69	14.1%	.88	14.9%	.70	14.0%	.74
	50	**8.1%**	**.85**	9.6%	.80	8.9%	.94	10.0%	.81	8.7%	.86
10%	100	6.6%	.83	**7.4%**	**.90**	9.4%	.86	10.0%	.75	9.1%	.81
	150	6.9%	.79	9.1%	.79	**8.2%**	**.98**	10.7%	.72	8.9%	.75
	500	6.3%	.83	8.6%	.83	9.0%	.93	**9.8%**	**.79**	**7.2%**	**.85**
	20	19.4%	.62	17.8%	.66	17.4%	.84	17.4%	.68	18.5%	.62
	50	12.4%	.75	14.2%	.71	14.1%	.91	13.6%	.79	12.7%	.80
20%	100	12.5%	.66	13.7%	.75	14.6%	.80	15.1%	.58	13.6%	.66
	150	11.6%	.67	12.8%	.79	15.0%	.68	14.3%	.57	13.3%	.65
	500	10.4%	.76	11.3%	.82	12.2%	.89	12.0%	.80	11.0%	.85

Although the match between the G/T–model and the data is not perfect, the

analysis of the results allows us to make the following qualitative conclusions:

1. Under–performance for low levels of stimulation may be explained by the model with the low goal value (G).

2. Improvement of performance with increase of the stimulation can be explained by increase of the goal value (G). This also corresponds to the idea that motivation of the subjects to choose the door correctly increased for stronger stimuli.

3. At extremely high levels of stimulation the noise (T) may also increase, which may explain degradation of performance in set I and III.

4.4 The S_p–Model

Successful performance of the model depends not only on learning new production rules and their statistical parameters, but also on the ability to retrieve these rules quickly, that is on strengths of the new production rules. If a rule has not been used for some period of time, then it can get 'forgotten'. This effect can be simulated in ACT–R using the production strength. The model takes into account time intervals between the tests as well as between the training days. The model described in this section uses the production strength learning mechanism to study the possible implications of strength learning and different decay rates.

4.4.1 Influence of Production Strength Learning

Production strength S_p affects the time necessary for a production rule to fire (see equation (2.4)). Low production strength results in longer latencies and higher costs. The production strength learning mechanism of ACT–R increases the strength of productions that are used more frequently. The speed of the production strength learning can be controlled by the `:sl` parameter or d in equation (2.8).

As was explained earlier the model is organised in such a way that only the new production rules learned during the model run can be significantly affected by the strength learning mechanism. Initially these new rules have low production strengths and, as a result, relatively higher costs. With practice or repetitive learning the strength may increase.

Figure 4.7 shows the dynamics of strengths (top) and costs (bottom) of the production rules learned during the model run. The graphs are shown for three settings of the

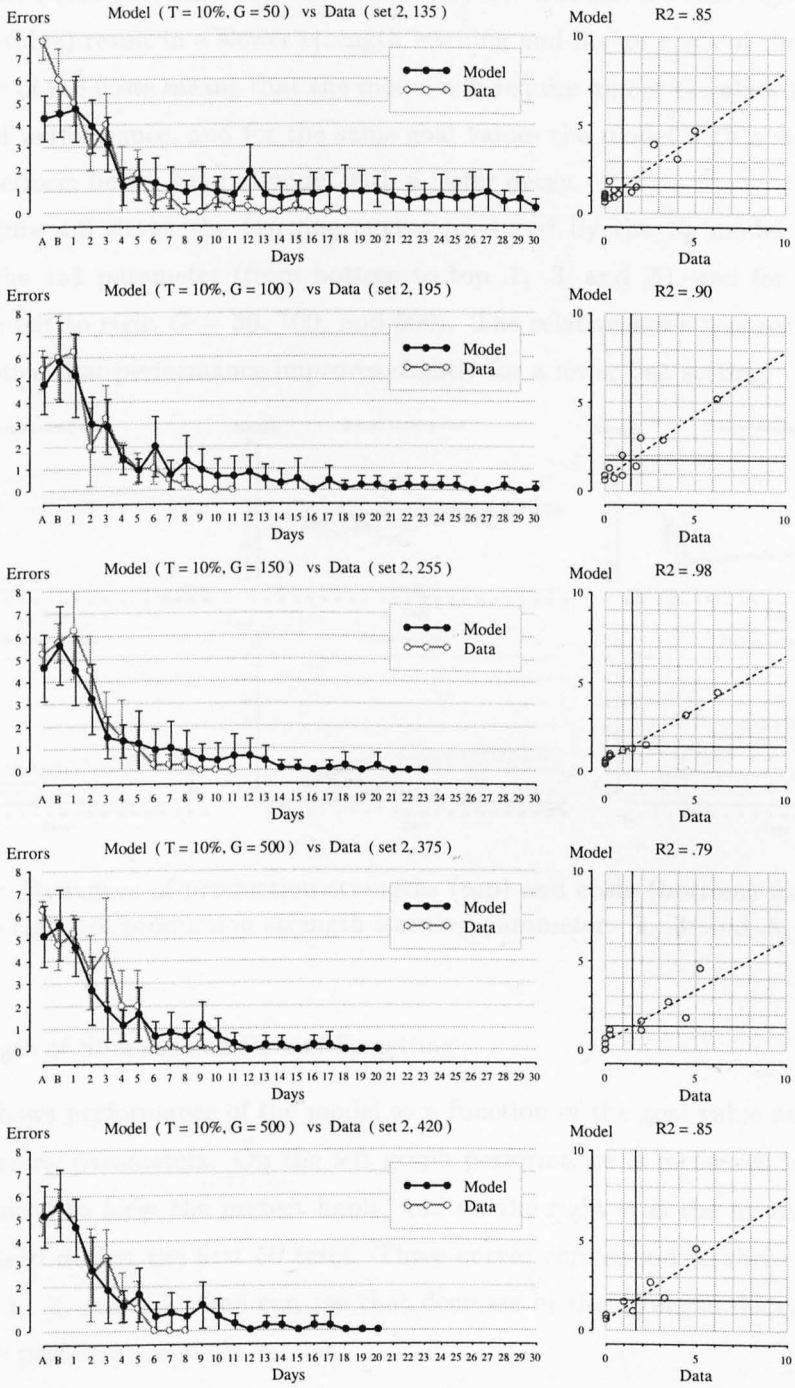

FIGURE 4.6: Comparison of G/T–model performance with data from the set 2 (good visual discrimination). Left: learning curves. Right: regression plots.

:sl parameter (from bottom to top sl = .1, .3, and .5). One can see that higher decay rates (higher :sl value) result in a slower strength learning and higher costs of the productions. The increase of the costs means that the model will require higher G values to achieve the same kind of performance, and for the same goal values the model with a slower strength decay will perform better than a model with a faster decay.

Figure 4.8 shows the learning curves produced by the S_p model for the three settings of the :sl parameter (from bottom to top .1, .3, and .5), and for different goal values (from left to right G = 50, 100, and 500). The relative noise was set to T = 10%. One may notice that performance improves slightly for a lower :sl setting.

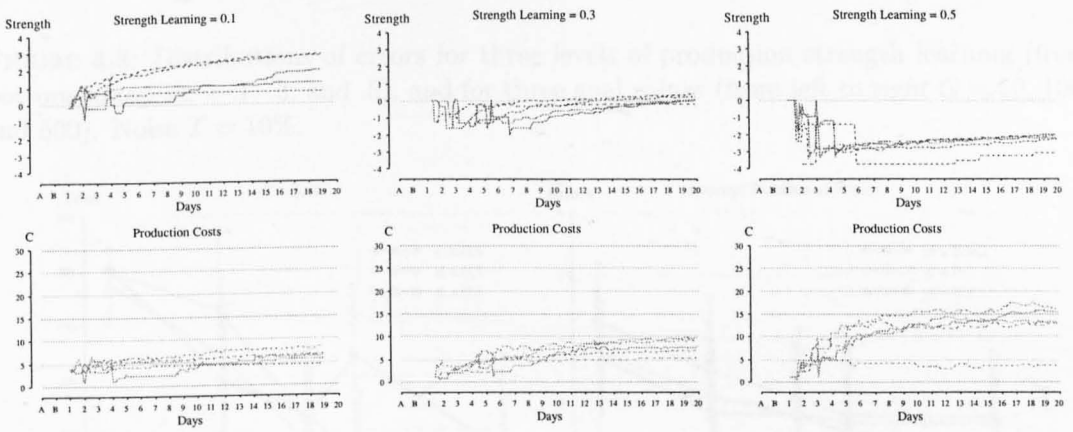

FIGURE 4.7: Dynamics of production strengths (top) and costs (bottom) for three values (from left to right) of production strength learning parameter: .1, .3, and .5.

4.4.2 Strength of Stimulus and Rate of Forgetting

Figure 4.9 shows performance of the model as a function of the goal value and production strength learning parameters. On the left graph performance is measured in the number of tests required to form the perfect habit, and on the right it is the average number of errors produced during the first 50 tests. Three curves correspond to three values of :sl parameter (.1, .3, and .5). One can see that decrease of the strength decay rate slightly improves the performance of the model.

One may speculate that performance improvement with the increase of stimulus is due to a lower decay rate of the knowledge learned under a stronger stimulus (e.g. due to a higher initial activation). There is evidence supporting the idea that memories formed

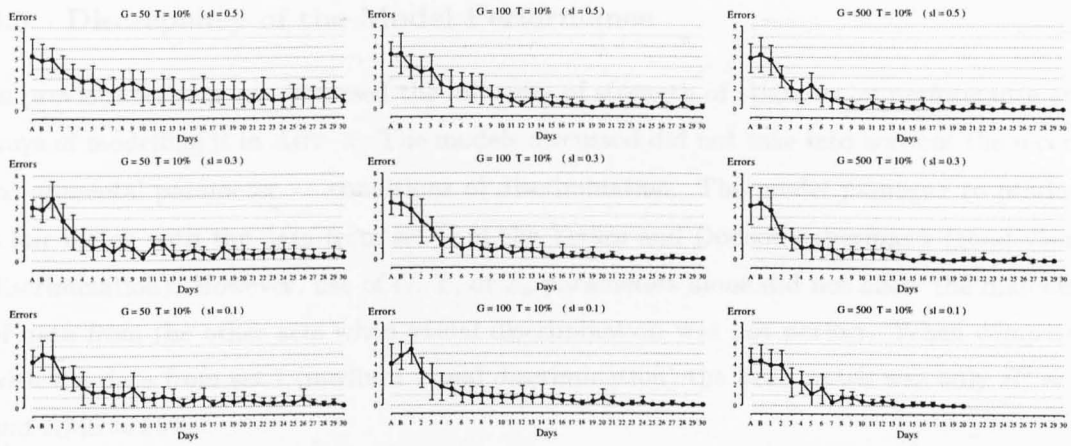

FIGURE 4.8: Distributions of errors for three levels of production strength learning (from bottom to top `sl` = .1, .3, and .5), and for three goal values (from left to right $G = 50$, 100, and 500). Noise $T = 10\%$.

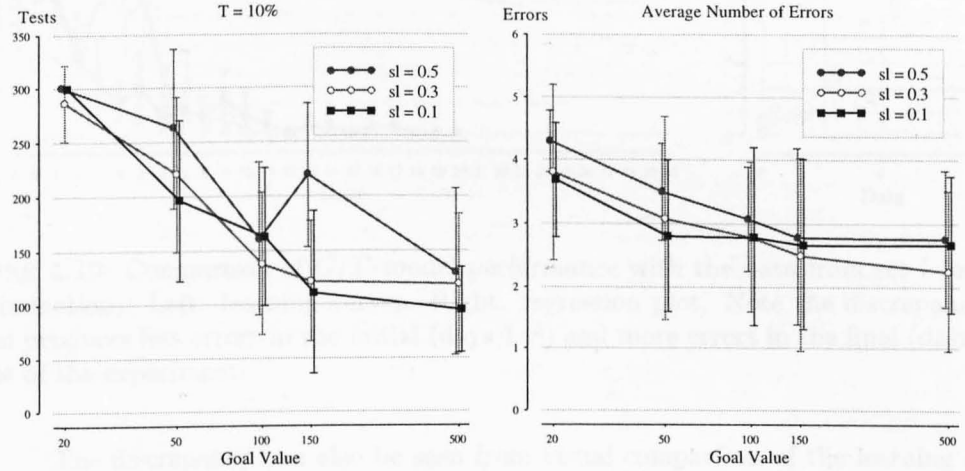

FIGURE 4.9: Performance as a function of the goal value and three values of strength learning parameter. Left: the number of tests needed by the model to form the perfect habit. Right: average number of errors produced during the first 50 tests.

in highly emotional states (high level of arousal) are stronger. This phenomenon is known as 'flashbulb' memories (Brown & Kulik, 1977). However, it is not clear whether this effect influences learning procedural knowledge (skills). If it does, then the model indicates that it can be modelled using production strength in ACT–R, and that it is more likely that the decay rate decreases under a stronger stimulation.

4.5 Discrepancy of the Model Performance

In previous sections we discussed the influence of strength of stimulus on performance and ways of modelling it in ACT–R. The models discussed did not take into account the second experimental parameter — conditions of discrimination. The model managed to produce a fair match with the data from set II of the Yerkes and Dodson experiment (good visual discrimination). However, use of G, T, or S_p parameters alone did not allow the matching of data from the other sets when visual discrimination was not perfect. When compared with the data from set I (medium visual discrimination) the best match was only $R^2 \approx .5$ and RMS $\approx 15\%$.

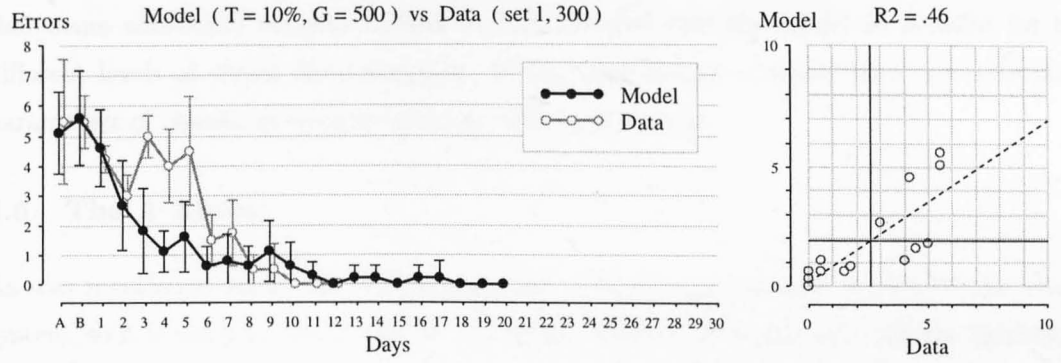

FIGURE 4.10: Comparison of G/T–model performance with the data from set I (medium discrimination): Left: learning curves. Right: regression plot. Note the discrepancy: the model produces less errors in the initial (days 1–8) and more errors in the final (days 9–20) stages of the experiment.

The discrepancy can also be seen from visual comparison of the learning curves. A typical example is shown on Figure 4.10. On the left plot data from set I (stimulation 300) is compared with the best matching G/T–model ($T = 10\%$, $G = 500$). One can see that the data curve has a different character: although there are more errors produced in the first stage, after the fifth day errors decay much faster. Perfect habit in the data is formed on day 8, while the model achieved the same performance only on day 11. Plot on the right of Figure 4.10 shows the corresponding mean–square regression. One can see that the mean number of errors for the model (horizontal line) is smaller than that of the data (vertical line). The coefficient of determination $R^2 = .46$. The model produces less errors in the initial stage of the run, but more errors in the final stage.

In fact, a similar pattern can be observed in the results of the G/T–model (see Figure 4.6). Indeed, the model overperforms in the initial stage of the experiment and underperforms in the final. In general the model forms the perfect habit later than the real mice. This could be overcome by reducing the noise, but then the match in the initial stage would be even worse.

One possible solution to resolve this issue could be a decaying noise in conflict resolution. The idea that noise can be dynamic and decay over time was suggested before (Belavkin, 2001, also proposed by Taatgen at the ACT–R workshop, 2001). The model with decaying noise will be introduced and discussed in the next chapter. However, the expected gain noise hardly has any relevance with the condition of visual discrimination. It is clear that some additional variable should be incorporated into the model to account for the different levels of visual discrimination. In the next section a model that uses activation parameters of chunks in working memory will be described.

4.6 The A–Model

As was mentioned earlier the model does not use any sophisticated models of the visual system, so it is not possible to achieve the desired effect by directly reducing the lighting in the simulated environment. However, it is possible to adjust the model parameters in order to simulate the effect. The mouse could learn the correct strategy only if it paid attention to the colour properties of the doors. With the darker conditions, however, the task became more difficult because the key feature (door colour) became less obvious.

In the model the correct learning happens when one of the two learning rules with colour chunks in the condition fires. If these learning rules fail to retrieve the chunks representing the colour of the door, then the rule will fail to fire altogether, and, as a result, the model will not learn the correct strategy. This means that we can use activation of the colour chunks in order to simulate different lighting conditions.

4.6.1 Influence of Activation Parameters

There are several factors in the model that are influenced by chunk's activation:

- Lower activation results in a longer retrieval time, and consequently longer latency time. This increases the cost of the production rule, thus reducing its chance to win in the conflict resolution process.

- If the activation of a chunk is below the retrieval threshold, then the rule may not fire at all.

Figure 4.11 illustrates the way activation parameters were used to simulate different discrimination conditions. The retrieval threshold was set to the default value of 0 (shown by the horizontal dotted line). The `:ans` parameter controlling the activation noise variance was set to .15. The activation base level (A) of the chunks *black* and *white* was used to control the mean of activation distribution. Figure 4.11 demonstrates from left to right the dynamics of colour chunks activations for three values of base level activation ($A = .5$, 0, and $-.2$).

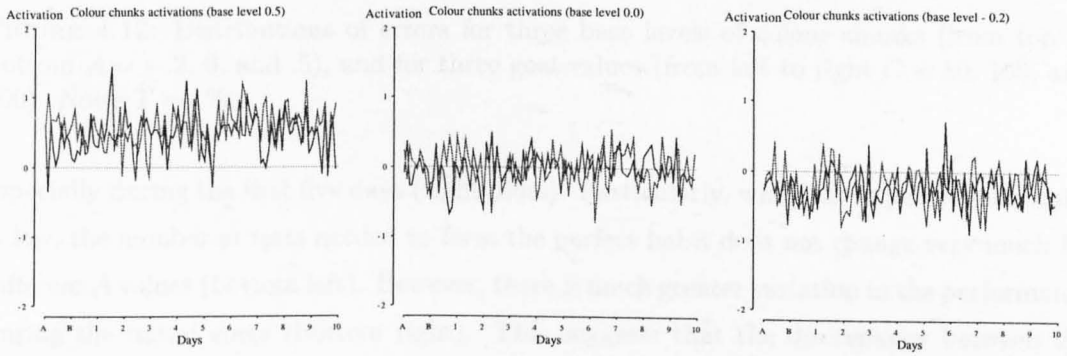

FIGURE 4.11: Different lighting conditions simulated using activations of colour chunks with different base levels (from left to right $A = .5$, 0, and $-.2$), and retrieval threshold at 0.

Figure 4.12 shows the learning curves produced by the model run at different activation base levels of colour chunks (from top to bottom $A = .5$, 0, and $-.2$) and three goal values (from left to right $G = 50$, 150, and 500, noise $T = 1\%$). Performance of the model improves significantly for higher values of A. The character of the learning curves corresponds better to the shapes of the learning curves from the first experimental set: the model produces more errors in the initial stage, but once the correct strategy has been learned, the errors decay much faster.

Figure 4.13 shows the performance curves (perfect habit on the left and errors during the first fifty tests on the right) as a function of goal value (G) and activation base level (A). The upper plots show performance of the model for $T = 10\%$, and the lower for $T = 1\%$. Three curves on each graph represent performance for three different values of A. One may see that activation of the colour chunks affects significantly the performance

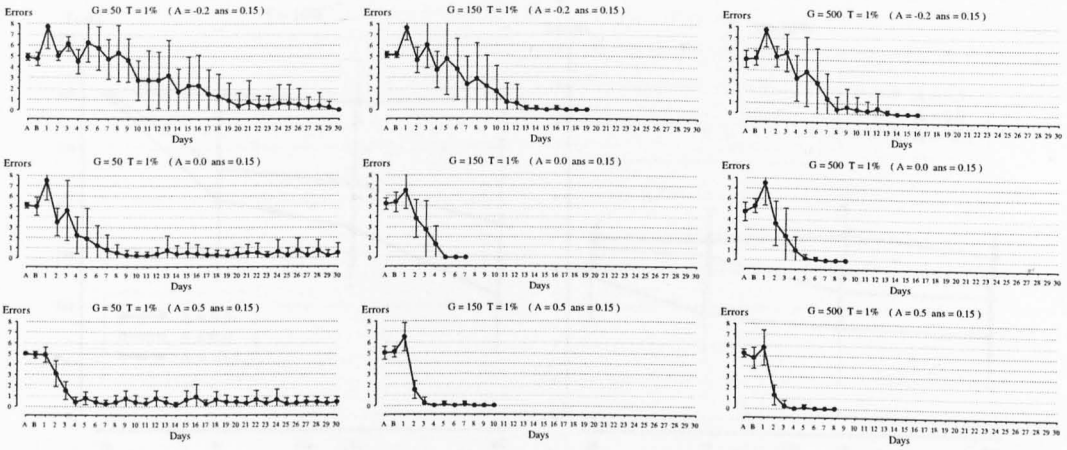

FIGURE 4.12: Distributions of errors for three base levels of colour chunks (from top to bottom $A = -.2$, 0, and .5), and for three goal values (from left to right $G = 50$, 150, and 500). Noise $T = 1\%$.

especially during the first five days (right plots). Particularly, when the expected gain noise is low, the number of tests needed to form the perfect habit does not change very much for different A values (bottom left). However, there is much greater variation in the performance during the initial stage (bottom right). This suggests that the discrepancy between the model and data shown on Figure 4.10 can be reduced by varying the A parameter of the model (activations of colour features of the doors).

4.6.2 Matching the A–Model to the Data of Set I and III

Results indicate that the A–model is indeed better suited for modelling the data from set I and III (medium and slight discrimination conditions). In particular, the A–model ran at $A = -.2$ produced a good match with the data, and it is significantly better than for the G/T–model. Table 4.3 reports R^2 and RMS for different settings of G and T for the A–model with $A = -.2$ compared with set I (medium visual discrimination).

By looking at Table 4.3 one may notice that the better matches are now located in the area of lower noise ($T = 1\%$, 5%). This can be explained by the fact that low activation of colour chunks reduced the performance of the model, thus the noise should be lower in order to compensate for the additional errors the model produces.

Figure 4.14 illustrates one–to–one comparison of the learning curves from set I with the best model results in terms described in Section 4.2 (shown in bold in Table 4.3). One

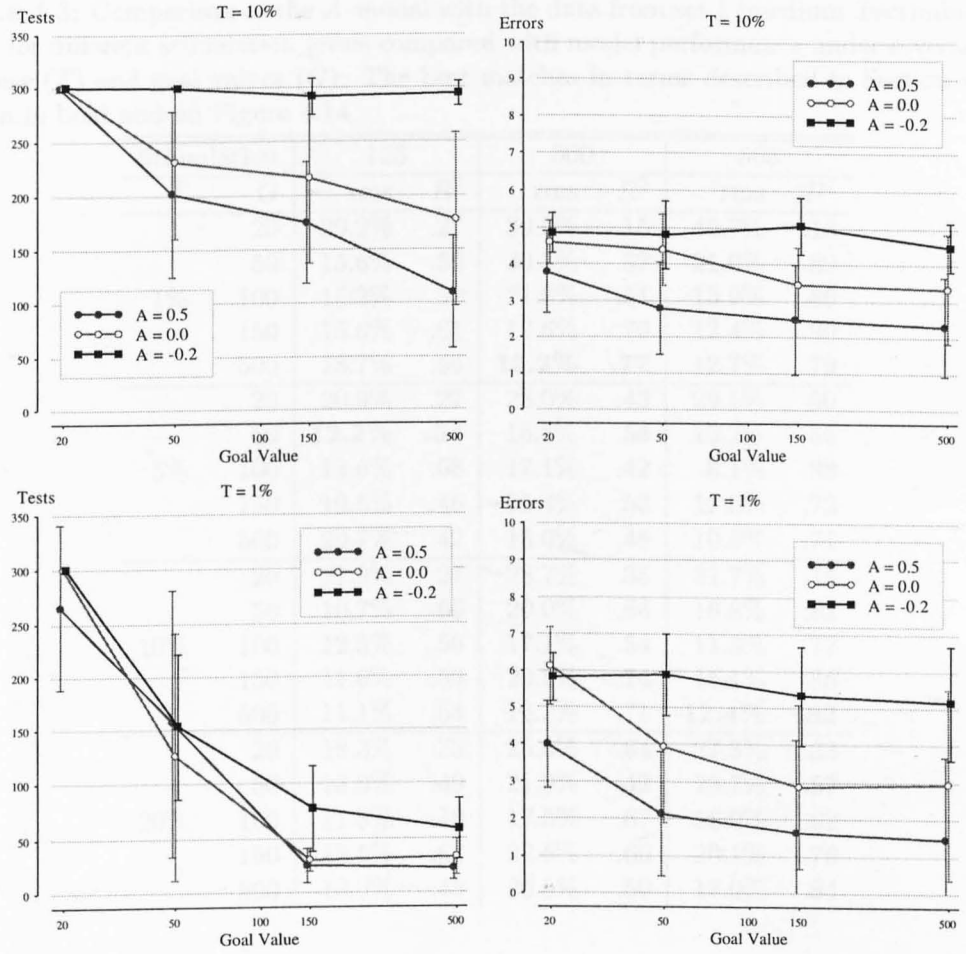

FIGURE 4.13: Performance as a function of the goal value parameter for three levels of relative noise T. Left: the number of tests needed by the model to form the perfect habit. Right: average number of errors produced during the first 50 tests.

can see that at a low level of stimulation the performance is not good, and in fact the model (and mice) did not form the perfect habit. The model reproduces this behaviour with $G = 50$ and $T = 5\%$. The best performance is at a moderate level of stimulus, which corresponds to the model with low noise ($T = 1\%$) and high goal values ($G = 150 \sim 500$). Further increase of the level of stimulus leads to performance degradation, which is reflected in the model by increase of noise ($T = 5\% \sim 10\%$). There is, however, a significant discrepancy with the data: the errors in the data decay faster than those of the model in the final stages of the experiment (after day 10). Note that a similar discrepancy was observed in the Tower of

TABLE 4.3: Comparison of the A–model with the data from set I (medium discrimination). Data for different stimulation levels compared with model performance under several levels of noise (T) and goal values (G). The best matches in terms described in Section 4.2 are shown in bold and on Figure 4.14.

Stimulation		125		300		500	
T	G	rms	R^2	rms	R^2	rms	R^2
	20	29.2%	.27	39.5%	.15	40.7%	.16
	50	15.6%	.59	30.3%	.57	21.0%	.89
1%	100	15.2%	.52	21.8%	.51	13.9%	.86
	150	15.6%	.61	16.6%	.76	12.4%	.90
	500	18.7%	.55	**13.2%**	**.77**	12.7%	.79
	20	20.9%	.27	28.0%	.43	29.1%	.50
	50	**12.2%**	**.54**	16.8%	.54	10.2%	.86
5%	100	13.6%	.58	17.1%	.42	8.1%	.88
	150	19.5%	.46	15.4%	.53	11.0%	.72
	500	20.7%	.42	16.0%	.46	10.5%	.71
	20	21.9%	.27	28.7%	.35	31.7%	.15
	50	10.7%	.66	20.0%	.64	16.8%	.82
10%	100	12.5%	.56	17.1%	.54	11.3%	.77
	150	11.0%	.59	20.5%	.74	15.1%	.86
	500	11.1%	.54	16.2%	.71	**12.4%**	**.82**
	20	18.3%	.33	25.0%	.64	27.3%	.33
	50	13.0%	.49	21.9%	.42	18.7%	.57
20%	100	11.8%	.49	17.5%	.67	16.0%	.62
	150	12.5%	.62	22.6%	.60	20.1%	.76
	500	13.0%	.45	20.5%	.50	17.0%	.84

Nottingham model, and a decrease of the expected gain noise variance towards the end of a task was proposed as a solution (Belavkin, 2001).

Set III (slight visual discrimination) is the most difficult to model because only two subjects were used to collect data for each setting. Thus, the experimental distribution is very approximate. Nevertheless the A–model was compared with the data of set III and managed to produce a fair match. Table 4.4 shows the results of this comparison and learning curves from the best matches in terms described in Section 4.2 are shown on Figure 4.15.

Again, the results indicate that performance at a low level of stimulus is reproduced better by the model with low goal value ($G = 50$). The best performance of subjects is at the moderate level of stimulation, which corresponds to the model with higher goal values ($G = 150 \sim 500$). Finally, high levels of stimulation hinder the performance, which

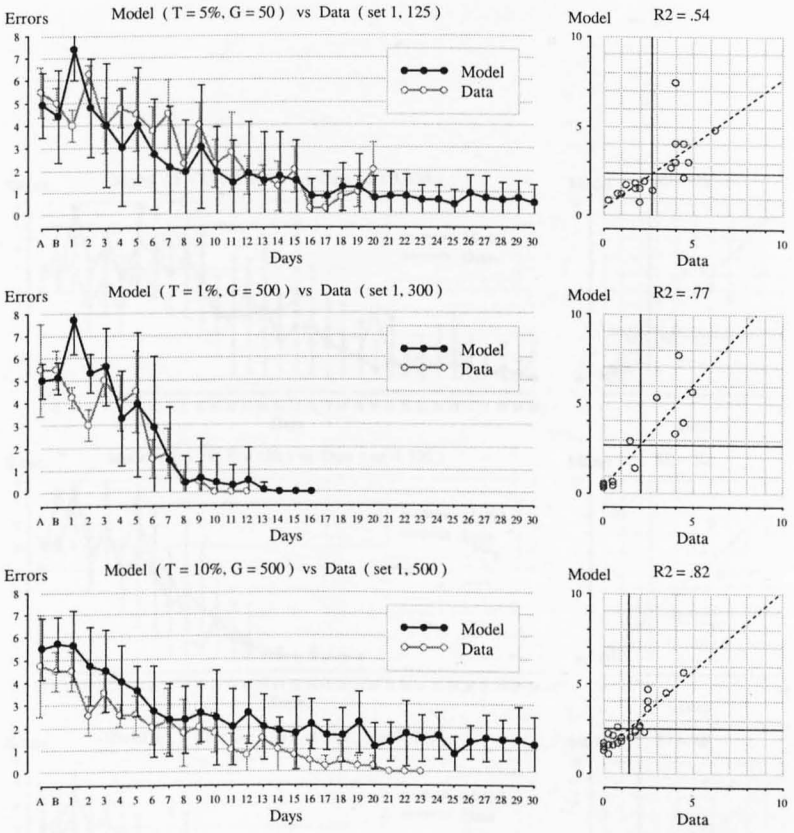

FIGURE 4.14: Comparison of the learning curves from set I (Yerkes & Dodson, 1908) with the A–model. Left: learning curves. Right: regression plots.

is reflected in the model by increase of noise ($T = 5\% \sim 10\%$). Again, the model fails to match well the decay of errors in the data in the final phase of the experiment (after day 10).

4.6.3 Reproducing the U–effect

Let us return to Figure 4.1 (adopted from Yerkes & Dodson, 1908) that shows a U–shaped dependencies of performance with respect to the strength of stimulus for medium and slight discrimination conditions (curves for set I and III). These experimental results first discovered by Yerkes and Dodson became the foundation for the inverted–U law. Having found the model settings that fit better the profiles of the experimental learning curves, we may now compare the performance of these models with the performance of subjects for

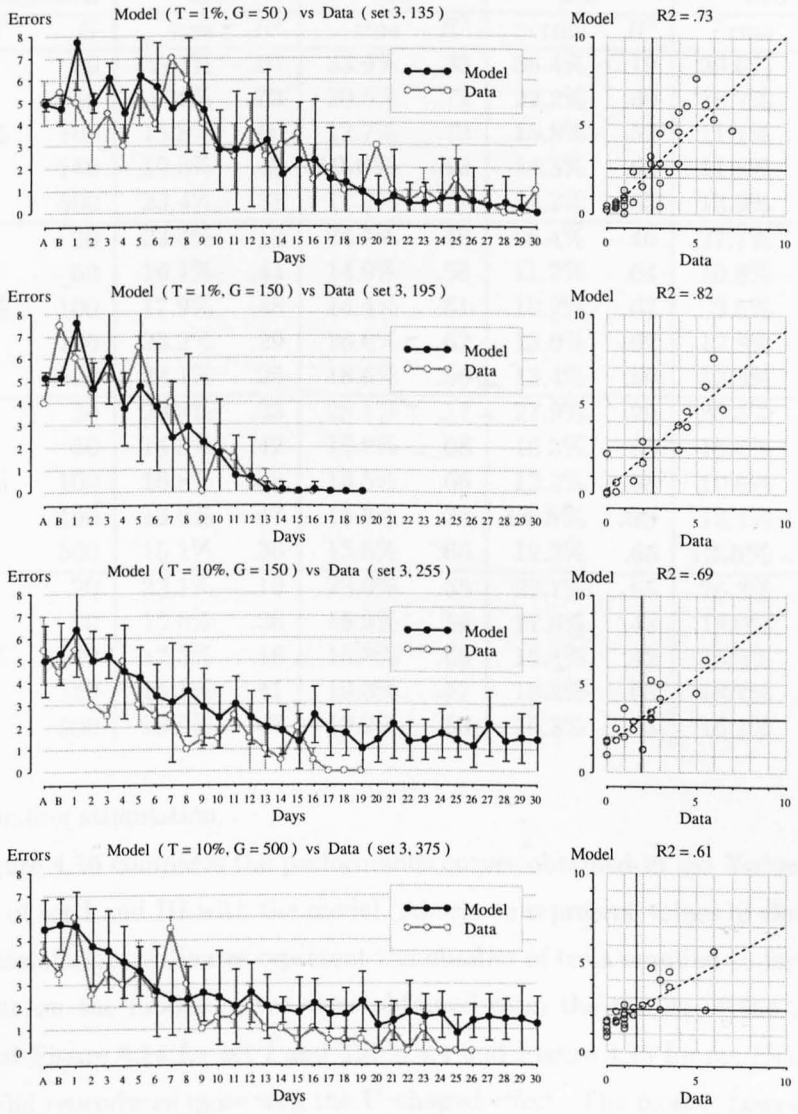

FIGURE 4.15: Comparison of the learning curves from set III (Yerkes & Dodson, 1908) with the A–model. Left: learning curves. Right: regression plots.

TABLE 4.4: Comparison of the A–model with the data from set III (slight discrimination). Data for different stimulation levels compared with model performance under several levels of noise (T) and goal values (G). The best matches in terms described in Section 4.2 are shown in bold and on Figure 4.15.

Stimulation		135		195		255		375	
T	G	rms	R^2	rms	R^2	rms	R^2	rms	R^2
	20	33.1%	.07	33.9%	.33	36.4%	.19	39.6%	.18
	50	**11.4%**	**.73**	20.5%	.72	22.2%	.60	16.7%	.75
1%	100	15.8%	.60	12.7%	.72	15.8%	.57	11.1%	.76
	150	19.3%	.49	**10.0%**	**.82**	14.3%	.64	11.8%	.72
	500	23.4%	.31	12.9%	.77	14.2%	.65	13.9%	.60
	20	23.4%	.12	24.7%	.48	26.4%	.40	27.7%	.35
	50	16.1%	.44	14.9%	.58	11.3%	.64	10.8%	.63
5%	100	17.9%	.48	14.4%	.61	10.2%	.67	9.5%	.70
	150	23.2%	.29	16.6%	.62	13.0%	.62	12.8%	.60
	500	24.1%	.25	18.6%	.56	13.4%	.59	13.4%	.56
	20	23.3%	.23	28.1%	.17	27.9%	.29	30.5%	.15
	50	14.3%	.42	16.9%	.68	16.3%	.60	15.4%	.63
10%	100	16.8%	.38	13.5%	.66	12.3%	.58	10.6%	.65
	150	13.6%	.47	15.3%	.74	**14.5%**	**.69**	13.1%	.72
	500	15.1%	.36	15.6%	.66	12.3%	.66	**12.5%**	**.61**
	20	22.1%	.10	23.0%	.55	22.7%	.65	26.3%	.36
	50	15.6%	.36	18.9%	.56	17.8%	.43	18.0%	.53
20%	100	17.3%	.16	15.8%	.65	15.4%	.42	15.9%	.45
	150	15.1%	.41	19.3%	.57	18.2%	.65	18.7%	.60
	500	13.9%	.47	19.2%	.53	16.3%	.63	16.1%	.65

the corresponding stimulation.

Figure 4.16 compares the performance curves obtained in the Yerkes and Dodson experiments of set I and III with the model. Abscissae represent values of electric stimulus in units of stimulation; ordinates represent the number of tests required to form the perfect habit. Points on the model curves were obtained from the results of the A–model (see Table 4.3 and Figure 4.14 for set I and Table 4.4 and Figure 4.15 for set III). One can see that the model reproduces quite well the U–shaped effect. The model, however, forms the perfect habit significantly later than mice at a high level of stimulation. It will be shown in the next chapter that introduction of a dynamic noise in the model enables to overcome this problem.

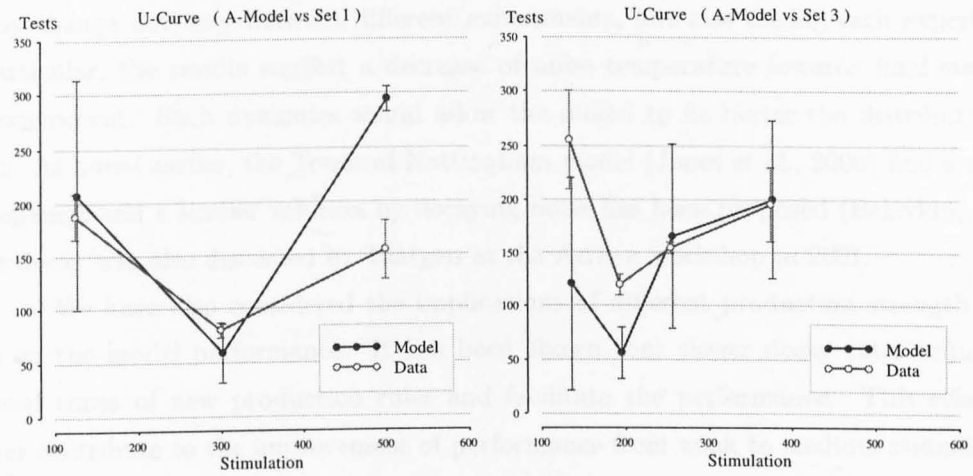

FIGURE 4.16: Comparison of the performance curves. Left: set I (medium discrimination) compared with the A–model results (see also Figure 4.14). Right: set III (slight discrimination) compared the performance of the A–model (Figure 4.15).

4.7 Conclusions

In this chapter the behaviour of the Dancer model has been studied with regards to manipulation of several parameters of the ACT–R cognitive architecture. It has been demonstrated that results of the Yerkes and Dodson experiment can be reproduced by modifying parameters of the conflict resolution mechanism, which is a model decision making. In particular, the strength of stimulus can be modelled by the goal value parameter G. The degradation of performance for exptermely high stimulation (inverted–U effect) can be modelled using two parameters:

- Increased relative noise T in conflict resolution.
- Further decrease of activations of chunks representing the door features. Note that Humphreys and Revelle (1984) suggested to explain the inverted–U effect by the reduced shot–term memory capacity for high arousal.

One might suggest that the variation in performance that mice demonstrated in these experiments was due to the changes in the decision making system under different conditions. The reasonable question here is whether such changes serve any useful function. Furthermore, analysis of the discrepancy between the model and data in the initial and the final stages of the experiments as well as the variances in data and model suggest that G and

T may change not only between different experiments, but also during each experiment. In particular, the results suggest a decrease of noise temperature towards final stages of the experiment. Such dynamics would allow the model to fit better the distributions of errors. As noted earlier, the Tower of Nottingham model (Jones et al., 2000) had a similar discrepancy, and a similar solution by decaying noise has been proposed (Belavkin, 2001). Noise decay was also discussed by Taatgen at the ACT–R workshop in 2001.

We have also considered the implications of different production strength decay rates on the model performance. It has been shown that slower decay rates reduce the retrieval times of new production rules and facilitate the performance. This effect can further contribute to the improvement of performance from weak to medium stimuli.

Although the manipulations of several parameters improved the model fit to data, the theory as it stands does not fully explain the values of some parameters. For example, why and how the noise temperature in conflict resolution should change? In the next chapter the role of the noise and its implication on learning will be analysed using entropy reduction in the system. It will be shown that dynamically changing G and T not only can improve the model performance, but also may optimise the learning process. The need for an alternative conflict resolution theory will be exposed.

CHAPTER 5

Uncertainty, Noise and Emotion

Noise in the ACT–R conflict resolution process was introduced to match the performance of subjects in probability matching experiments (Anderson, 1993; Lovett & Anderson, 1995). It is considered mainly as an artifact necessary to overcome the limitations of ACT–R cognitive models and to account for the stochastic nature of the human brain. Indeed, because cognitive models are usually have very few rules as opposed to human brain, without noise the behaviour of ACT–R models would be just too deterministic and predictable. The nature of the noise is addressed mainly to the complexity of the human brain. But is this its only useful application?

The Dancer model discussed in previous chapters demonstrated that noise and goal value can be used to model the different behaviour under different levels of stimulation (and arousal). The level of arousal is regulated by the autonomic nervous system through various mediators such as hormones and neurotransmitters. The conscious awareness of these states is attributed by many psychologists to primary emotions. Fear, for example, is a very basic reaction initiated in the *amygdala* causing adrenaline release into a blood preparing the body for a hazard detected in the environment (LeDoux, 1990). The amygdala also sends signals to other parts of the brain, but the exact purpose of these transmissions is not yet fully understood. The noise increase in the model simulating the behaviour at extremely high levels of stimulus could be explained by the intrusion of the autonomic nervous system into the decision making system of the brain. Similar interactions are known between other parts of the brain and occurring during the experiences of other primary emotions. For example, dopamine and serotonine transmissions are known to be active particularly during the experiences of joy and distress. It is possible to imagine that the character of decision making changes during the experience of these emotions as well.

Analysis of the results of the Dancer model suggested that the conflict resolution parameters should be dynamically modified. For example, it is more likely that noise gradually decays during the experiment. Indeed, if the noise is somehow related to the experience of fear and flee reaction after receiving an electrical shock, then a decrease of errors during the experiment would reduce the number of such experiences, thus making decision making less noisy and more certain. Noise decay was suggested to improve the performance of other cognitive models (Belavkin, 2001, or Taatgen in his talk at the ACT–R workshop 2001).

In this chapter the role of the noise in learning and problem solving will be considered in a greater detail. It will be shown that noise decay may correspond to uncertainty reduction during the learning process, and that noise decay optimises the learning process in a way similar to simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983). It will be shown also that dynamic control of the goal value may further improve the learning process and performance of a problem solver. We speculate in the end of this chapter about the role of emotion and motivation on problem solving and intelligence due to their effect on decision making similar to some optimisation techniques.

5.1 Noise and Uncertainty

Let us consider an ACT–R model from a machine learning point of view. The model is to learn and solve a problem trying to achieve maximum performance and efficiency. It was shown that an increase of noise in conflict resolution leads to a degradation of performance on a task. The noise, however, is necessary to choose at random production rules when all the options have equal expected gains. For example, when beginning to solve a problem for the first time, there is no information about the expected probabilities or costs. Hence, all the expected gains are equal. But once the information has been learned from the experience should not the noise be removed?

The expected probability P_i in the ACT–R utility equation (2.1) represents only the past experience. If a rule was successful in all previous applications, however, such that $P_i = 1$, it does not guarantee that the rule will lead to a success next time it is applied. The rule may be missing some condition or the environment may change, and the previously successful rule may lead to a failure. For example, in the Yerkes and Dodson task the model may learn a rule to avoid the door on the left, and it can work fine for some time until

the order of the doors is changed. Without the noise the model will have to fail using the wrong rule at least as many times as it succeeded before, in order to equalise its chance with the other production rules in the system. Such learning by mistakes is not the best strategy particularly if the price of an error is high. Noise in the conflict resolution, however, increases the chance that another rule will fire, and, if it is successful, the model may learn by successes of another rule.

The example above illustrates that noise in conflict resolution is useful if the environment is changing. The possibility of the environment change imposes an uncertainty on the knowledge acquired during the past experience (i.e. the uncertainty of a success of decisions based on this knowledge). The information gained reduces this uncertainty, but it never ceases. One may view the noise in conflict resolution as a reflection of the uncertainty (or entropy) in the environment. If the uncertainty could be measured or estimated, then one could also define how significant the effect of noise (randomness) on the decision making process should be. That is if no information about the environment is available (high entropy), then the decision making should be completely random (high noise). On the contrary, if there is no uncertainty (zero entropy), then the decision making should be completely deterministic (no noise).

The entropy H of a system with states ξ is defined as

$$H\{\xi\} = -E\{\ln P(\xi)\} = -\sum_{\xi} P(\xi) \ln P(\xi)\,, \tag{5.1}$$

where $E\{\cdot\}$ denotes the expected value, and $P(\xi)$ is the probability of state ξ. It is quite difficult to estimate the entropy of a large system (e.g. the environment) with many states. We may, however, consider the problem from a different perspective. Let us consider a problem solver with n decisions (e.g. production rules implemented in ACT–R), and let $\xi \in \{0,1\}$ be a set of two states of the problem solver: failure ($\xi = 0$) and success ($\xi = 1$). Now we can estimate the uncertainty that the problem solver achieves the success state ($\xi = 1$). Let us call H_{01} calculated over just two states an *entropy of success*:

$$H_{01} = -\left[P(0) \ln P(0) + P(1) \ln P(1)\right]\,, \tag{5.2}$$

where $P(1)$ and $P(0)$ are probabilities of success and failure respectively (note that $P(0) = 1 - P(1)$).

The probability of success can be written as

$$P(1) = \sum_i P(1,i) = \sum_i P(1 \mid i)P(i)\,,$$

where $P(1,i)$ is the joint probability of event 1 and ith rule, $P(1 \mid i)$ is the conditional probability of 1 given that ith rule has fired, and $P(i)$ is the probability that ith rule fires. Note that for ACT–R $P(i)$ is the choice probability given by the Boltzmann equation (2.9).

The conditional probability $P(1 \mid i)$ is not known. ACT–R, however, records the information about the successes and failures of each rule in a form of expected (or empirical) probabilities P_i attached to each production rule (see equation (2.5)). Because the tests of each rule are independent, the empirical probability (2.5) asymptotically converges to the conditional probability $P(1 \mid i)$.[1] So, we can use ACT–R expected probabilities to calculate $P(1)$:

$$P(1) \approx \sum_i P_i\, P(i)\,.$$

The probability of failure is simply $P(0) = 1 - P(1)$. Now using expected probabilities P_i and calculating $P(i)$ by equation (2.9) we can estimate the probability of success:

$$P(1) = \frac{1}{\sum_i e^{E_i/\tau}} \sum_i P_i\, e^{E_i/\tau}\,.$$

Therefore, the entropy H_{01} of success in achieving the goal can now be calculated by equation (5.2).

Note that it is more convenient to use the following formula for calculating the empirical probabilities P_i:

$$P_i = \frac{\text{Successes}_i}{\text{Successes}_i + \text{Failures}_i + 1}\,.$$

This is because by default ACT–R sets initially all the probabilities $P_i = 1$ (the number of successes is set to 1 initially). Anderson and Lebiere justify this in order to make the prospects of a new production optimistic (Anderson & Lebiere, 1998, p. 135). This approach, however, is biased and not very convenient for calculating the entropy. Indeed, if at the beginning all $P_i = 1$, then the uncertainty of a success $H_{01} = 0$, which contradicts the idea that initial state should be the maximum entropy state (no experience). The above formula for the empirical probability is more suitable to estimate the entropy: it makes the initial values $P_i = 1/2$, which corresponds to the maximum entropy.

[1] It will be shown in Chapter 6 that this probability can be expressed through goal value G and cost C.

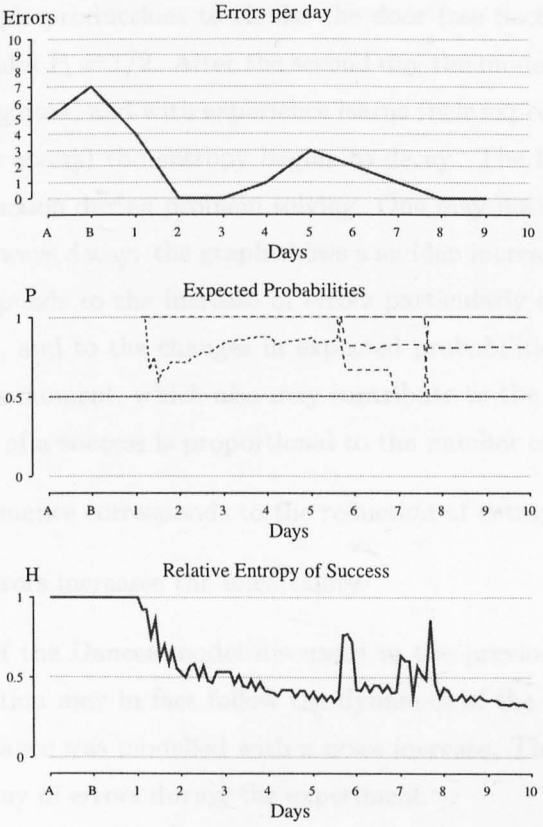

FIGURE 5.1: Dynamics of relative entropy of success during problem solving: error curve (top), probability learning (middle), and entropy change (bottom).

Because the number n of production rules may change due to learning new rules, it is convenient to use the relative entropy:

$$H_{\text{rel}} = \frac{H_{01}}{H_{\max}},$$

where $H_{\max} = \ln 2$ is the maximum entropy, when two states (success or failure) are equally probable (i.e. $P(1) = P(0) = 1/2$).

Figure 5.1 illustrates an example of the dynamics of the uncertainty of a success during problem solving. The top plot shows an error curve produced by the Dancer model. The plot in the middle shows traces of expected probabilities of the productions making the choice of a door (see Chapter 3). The lower plot illustrates the corresponding entropy of a success. As can be seen from the graphs the entropy is at its maximum in the beginning of the task. This is because during the first two days the dancer can escape through any

door and uses two simple productions to choose the door (see Section 3.3). The expected probabilities of these rules $P_i = 1/2$. After the second day the model learns new production rules to avoid the wrong door, and with experience learns their expected probabilities. With more successes (or less errors) the entropy begins to decay. The lower plot of Figure 5.1 shows the entropy reduction during problem solving. One may notice from the graph, that the entropy does not always decay: the graph shows a sudden increase of the entropy during day 5 and 7. It corresponds to the increase of errors particularly during day 5, as can be seen from the top plot, and to the changes in expected probabilities. In addition, ACT–R learns new rules at this moment, which also may contribute to the increase of uncertainty. In general the entropy of a success is proportional to the number of errors:

- Successful performance corresponds to the reduction of entropy.
- An increase of errors increases the uncertainty.

The results of the Dancer model discussed in the previous chapter suggest that noise in conflict resolution may in fact follow the dynamics of the entropy. Indeed, degradation of task performance was modelled with a noise increase. The suggested noise decay corresponds to the decay of errors during the experiment.

5.2 The H–Model with Noise Decay

Before we proceed to the analysis of the implications of dynamic conflict resolution parameters on learning, let us test the idea of entropy–regulated noise on the working model. The aim is to see what effect it produces on the model performance and whether the match with the data can be improved.

The H–model (H for entropy) is a modified A–model, described in the previous chapter. In this model relative noise T is dynamically controlled by the entropy parameter:

$$T(t) = T_0 H_{\text{rel}}(t)\,, \tag{5.3}$$

where t is time, $T_0 = T(0)$ is the initial value of the noise, $H_{\text{rel}}(t)$ is relative entropy of success for the task–related productions. In the Yerkes and Dodson experiment the task–related productions are rules that make the choice of the door.

The results of this model ran at different initial noise T_0 settings and for different goal values were compared with the data of sets I and III from Yerkes & Dodson, 1908. The

TABLE 5.1: Comparison of the H–model with decaying noise and the data of set I (medium discrimination). Data for different stimulation levels compared with model performance under levels of noise (T) and goal values (G). The best matches in terms described in Section 4.2 are shown in bold and on Figure 5.2.

Stimulation		125		300		500	
T_0	G	rms	R^2	rms	R^2	rms	R^2
	20	20.1%	.38	32.5%	.25	27.5%	.64
	50	18.7%	.59	16.2%	.83	16.8%	.68
1%	100	19.0%	.55	11.4%	.94	14.0%	.78
	150	16.6%	.60	7.7%	.84	7.0%	.85
	500	20.3%	.51	11.1%	.79	12.2%	.72
	20	13.2%	.48	18.7%	.66	17.2%	.65
	50	14.4%	.60	11.1%	.76	8.6%	.79
5%	100	21.1%	.39	14.3%	.71	12.8%	.74
	150	17.1%	.59	12.0%	.65	8.9%	.80
	500	14.2%	.75	**8.8%**	**.86**	7.9%	.85
	20	13.9%	.33	19.5%	.41	17.8%	.58
	50	11.9%	.49	15.3%	.62	9.8%	.85
10%	100	12.5%	.65	10.1%	.83	6.5%	.85
	150	14.0%	.46	13.0%	.68	9.7%	.75
	500	12.1%	.69	11.6%	.81	**7.1%**	**.88**
	20	13.0%	.51	20.1%	.48	16.8%	.62
	50	11.2%	.61	12.4%	.83	10.4%	.74
20%	100	**10.1%**	**.64**	16.9%	.69	10.8%	.83
	150	11.2%	.61	13.2%	.75	10.3%	.66
	500	9.8%	.72	15.8%	.57	8.7%	.85

analysis is presented in Tables 5.1 and 5.2. One can notice that the model can start now with the higher noise settings as opposed to the A–model, because noise is decaying. The best matches in the sense described in Section 4.2 are shown in bold. The learning curves of the selected models are compared with the experimental curves on Figures 5.2 and 5.3.

Firstly, the match between the model and the data in the mean points has improved (see Tables 4.3, 4.4 and Figures 4.14, 4.15). Secondly, and more importantly, the variances of the errors distributions correspond to each other much better. The improvement of model performance due to the dynamic noise control is illustrated in Table 5.3 by comparing correlations R^2 and RMS errors of the two models: a model with static noise (left) and the dynamic noise model (right). One can see that introduction of dynamic noise control into the model increased R^2 and reduced RMS practically in every case.

Figure 5.4 illustrates the U–shaped performance curves of the H–model compared

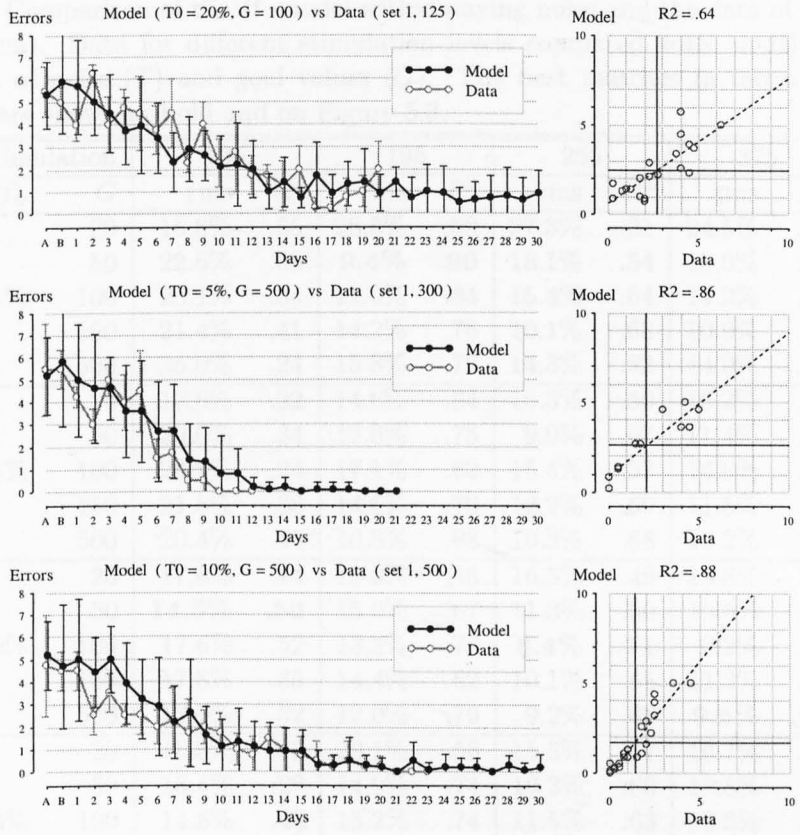

FIGURE 5.2: Comparison of the results of model with dynamic noise with the data set I (medium visual discrimination). Left: learning curves. Right: regression plots.

with the data of set I and III. Again, as predicted, the model fits data much better than a model with static noise settings (see Figure 4.16). These results illustrate that dynamic control of the expected gain noise using equation (5.3) can indeed improve the fit between the model and data.

5.3 More Noise, More Experience, More Information

Redundancy, defined as $1 - H_{\text{rel}}$, can be a good estimator of the information accumulated by a system. So, we can use the reduction in entropy to test how well the model learns under different parameter settings.

Because the choice probability $P(i)$ in ACT–R depends on the learned information and noise temperature τ (see equation (2.9)), the entropy of success H_{01} is not convenient

TABLE 5.2: Comparison of the H–model with decaying noise and the data of set III (slight discrimination). Data for different stimulation levels compared with model performance under levels of noise (T) and goal values (G). The best matches in terms described in Section 4.2 are shown in bold and on Figure 5.3.

Stimulation		135		195		255		375	
T_0	G	rms	R^2	rms	R^2	rms	R^2	rms	R^2
	20	15.6%	.65	26.5%	.59	27.3%	.31	24.5%	.58
	50	22.6%	.38	**9.4%**	**.90**	18.1%	.54	15.9%	.57
1%	100	23.1%	.34	11.0%	.84	15.4%	.64	14.2%	.62
	150	21.4%	.41	14.2%	.76	10.1%	.68	10.9%	.67
	500	25.0%	.24	15.8%	.72	14.3%	.62	14.9%	.53
	20	17.6%	.22	14.6%	.64	16.3%	.50	16.2%	.48
	50	20.0%	.34	12.6%	.75	9.9%	.67	11.4%	.57
5%	100	24.2%	.24	17.1%	.62	15.4%	.57	14.8%	.53
	150	21.8%	.38	14.6%	.70	12.2%	.60	11.5%	.65
	500	20.4%	.46	10.8%	.88	10.3%	.68	10.2%	.71
	20	17.6%	.14	19.3%	.46	16.5%	.49	18.8%	.36
	50	**14.5%**	**.50**	15.0%	.67	11.3%	.60	10.6%	.64
10%	100	17.6%	.52	13.3%	.76	**8.4%**	**.71**	9.2%	.68
	150	17.8%	.35	14.4%	.62	10.1%	.65	11.3%	.55
	500	17.7%	.52	12.0%	.79	9.2%	.68	**9.5%**	**.68**
	20	15.3%	.42	20.1%	.56	15.5%	.57	16.7%	.57
	50	16.4%	.29	14.9%	.74	10.3%	.66	11.8%	.59
20%	100	14.8%	.48	13.2%	.74	11.5%	.63	10.3%	.68
	150	16.3%	.35	14.9%	.69	8.3%	.79	11.4%	.56
	500	15.4%	.47	13.5%	.73	9.3%	.70	9.8%	.67

for estimating the learning in the system under different noise settings. In order to make this estimation independent of the conflict resolution mechanism let us assume that the choice of a rule is completely random $P(i) = \frac{1}{n}$. In this case probability of a success $P(1)$ will depend only on the learned empirical probabilities:

$$P(1) = \frac{1}{n} \sum_i P_i \, .$$

The entropy associated with this probability (calculated similarly by equation 5.2) can be used to estimate the knowledge accumulated in the system in the form of empirical probabilities P_i, because it is independent of the way the decisions are made. We shall refer to it as the *entropy of knowledge* H_k.

Although the noise may seem to hinder the performance, it in fact helps the learning process. Figure 5.5 illustrates the probability learning in the Dancer model for

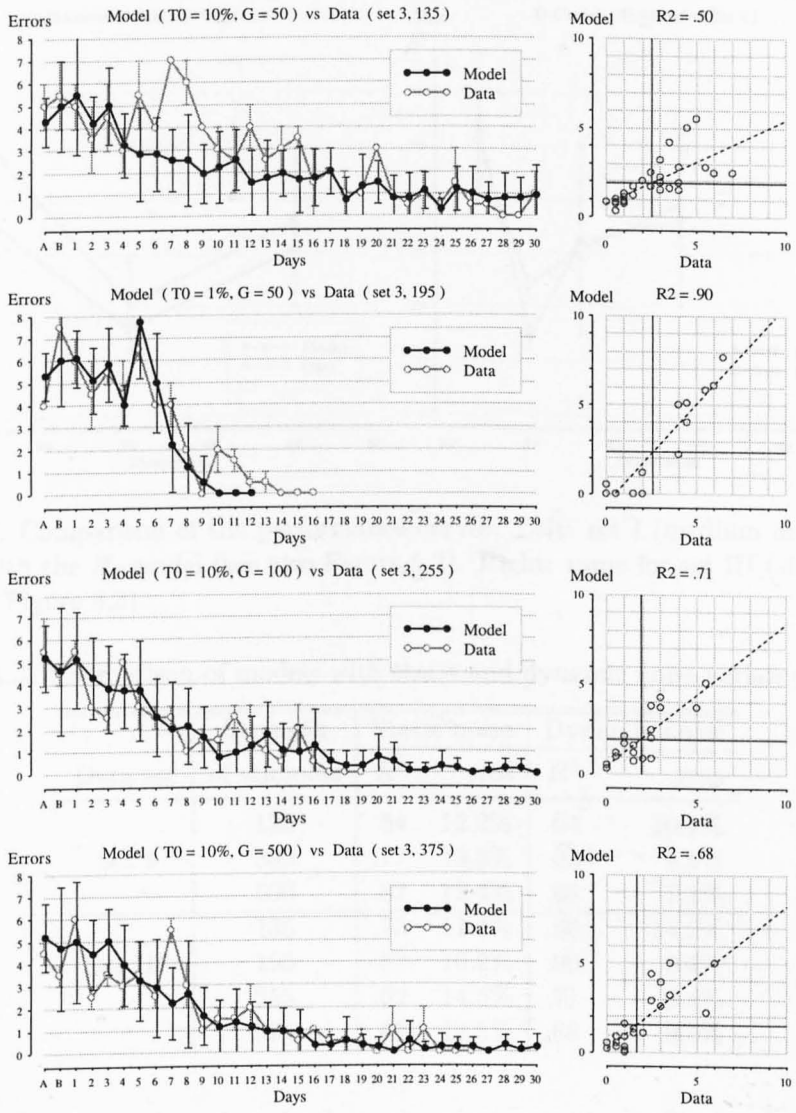

FIGURE 5.3: Comparison of the results of model with decaying noise with the data from set III (slight visual discrimination). Left: learning curves. Right: regression plots.

different noise settings. The left plot illustrates a trace of Ps of production rules during 110 tests with noise $T = 1\%$, and the right plot for $T = 20\%$. One may see that probabilities on the right plot were updated much more often, thus have more correct values. The entropy related to these probabilities is shown on Figure 5.6. One may see that by day 10 the entropy on the right plot decayed significantly more than on the left plot. Thus, by day 10 the model with a greater noise gained more information than the model with less noise.

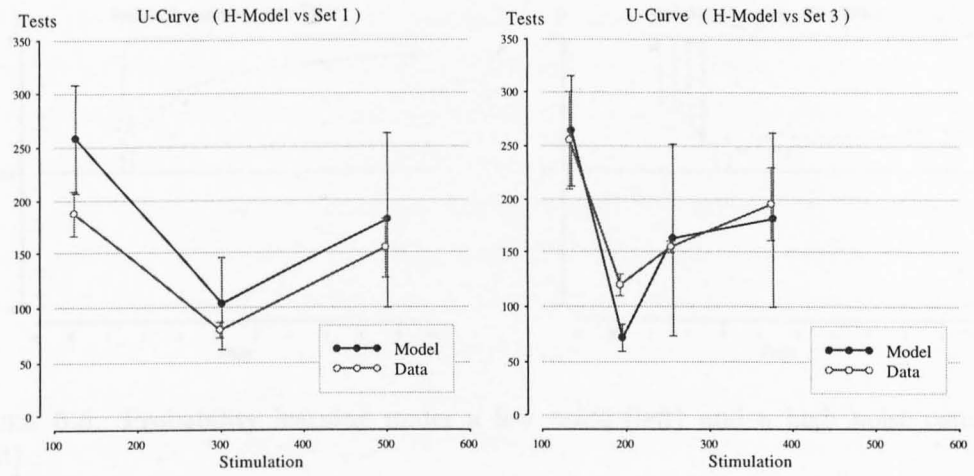

FIGURE 5.4: Comparison of the performance curves. Left: set I (medium discrimination) compared with the H–model (see also Figure 5.2). Right: same for set III (slight discrimination) (see Figure 5.3).

TABLE 5.3: Comparison of models with static and dynamic noise variance to data.

	Strength	Static noise		Dynamic noise	
Data set	of stimulus	R^2	rms	R^2	rms
Set I	125	.54	12.2%	.64	10.1%
	300	.77	13.2%	.86	8.8%
	500	.82	12.4%	.88	7.1%
Set III	135	.73	11.4%	.50	14.5%
	195	.82	10.2%	.90	9.4%
	255	.69	14.5%	.71	8.4%
	375	.61	12.5%	.68	9.5%

It is becoming clear that the dynamics of noise in the conflict resolution following the entropy can have a very useful purpose:

1. Noisy behaviour in the beginning of problem exploration supports gaining information about the task or the environment more quickly.

2. After the important information has been acquired, the reduction of noise narrows the search and encourages concentration on more successful decisions. If the learned knowledge is correct for the task or the environment, then keeping the noise low should improve performance.

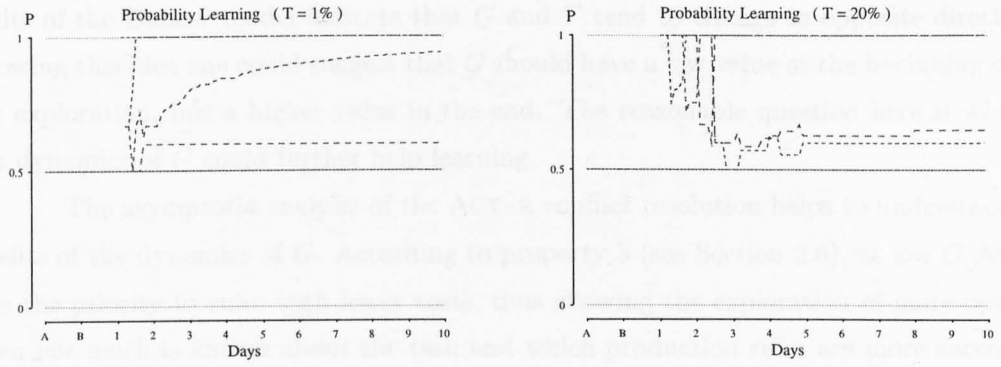

FIGURE 5.5: Probability learning under a low noise (left) and a high noise conditions (right).

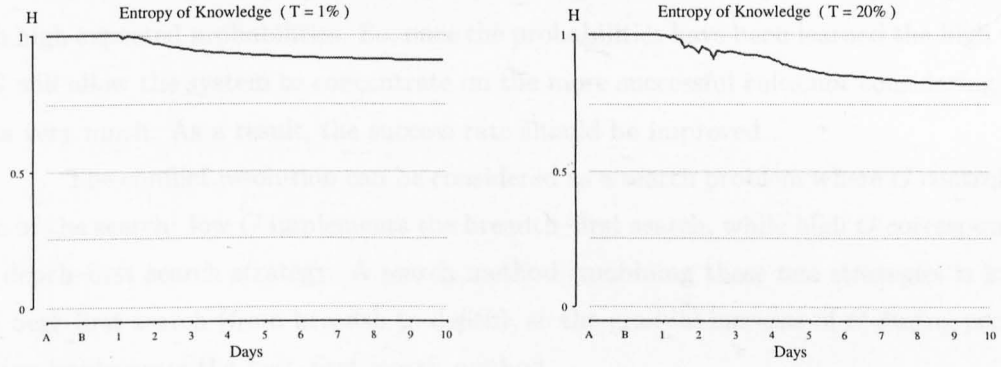

FIGURE 5.6: Dynamics of entropy under a low noise (left) and a high noise conditions (right).

3. If the environment changes and the number of errors suddenly increases, then a noise increase widens the search and allows the speeding–up of the learning process again.

5.4 Dynamics of Motivation

Noise is not the only parameter in the conflict resolution that can optimise the learning process. It was shown in the previous chapter that performance of the model is also very sensitive to the goal value (G). It has been known that cognitive performance depends on the motivation to achieve the goal (Atkinson, 1957, 1974). Theories of achievement motivation (e.g. Atkinson, 1974; Revelle & Michaels, 1976) support the idea that motivation is a function of probability of success, and hence it may change with experience. Moreover, the

results of the Dancer model indicate that G and T tend to change in opposite directions. Following this idea one could suggest that G should have a low value at the beginning of the task exploration, but a higher value in the end. The reasonable question here is whether such dynamics of G could further help learning.

The asymptotic analysis of the ACT–R conflict resolution helps to understand the benefits of the dynamics of G. According to property 3 (see Section 2.6), at low G ACT–R gives the priority to rules with lower costs, thus allowing the exploration of more options. When not much is known about the task and which production rules are more successful, such behaviour is a better strategy. However, if the 'cheap' productions do not achieve the success, then an increase of G should allow exploration of the more expensive options. Moreover, according to asymptotic property 4 at higher G ACT–R gives the priority to rules with high expected probabilities. So, once the probabilities have been learned the high value of G will allow the system to concentrate on the more successful rules not considering their costs very much. As a result, the success rate should be improved.

The conflict resolution can be considered as a search problem where G controls the type of the search: low G implements the breadth–first search, while high G corresponds to the depth–first search strategy. A search method combining these two strategies is known as a best–first search (from breadth to depth), so the gradual increase of G during problem solving implements the best–first search method.

5.5 Annealing Analogy

The above described dynamics of the conflict resolution parameters G and T implements yet another well–known heuristics — the optimisation by simulated annealing (Kirkpatrick et al., 1983). By looking at the 'soft–max' Boltzmann equation (2.9) for the choice probability in ACT–R (probability that ith rule will be selected in the conflict set), one may consider the expected gain $E = PG - C$ as a negative energy, and noise temperature τ as the temperature of the system. According to simulated annealing the following optimisation strategy applies:

- Problem solving should begin in a high temperature, high energy state, which implies low goal value G and high noise τ. This makes the exploration process more random. Note that such behaviour corresponds well to the exploration stage of problem solving in subjects. For example, when children start to tackle the Tower of Nottingham

puzzle, they are just playing with blocks and trying some simple constructions.

- During problem solving the system should cool down (annealing schedule defines the speed of cooling). This corresponds to an increase of G and decay of τ. In ACT–R it should make the choice less random and more dependent on the learned information. Note that when more information becomes known about the task, subjects express more confidence.

- In order to avoid local maxima (glass state) the system should melt up. This corresponds to a noise increase and goal value decrease. A similar situation in ACT–R model is illustrated on Figure 5.1: entropy increases on day 6 when additional errors occur. The entropy increase should be reflected in the conflict resolution by noise increase and decrease of the goal value.

- The crystallisation state corresponds, perhaps, to the goal state in problem solving, or in case of multiple solutions to a state when the perfect solution path has been discovered. In this case $\tau \rightarrow 0$.

5.6 ACT–R and a Two–Dimensional Model of Emotions

When humans encounter a success or a failure, a change in the environment, an impasse, or reach the final goal in a problem, they also experience emotions such as joy, frustration, surprise, anxiety, or excitement. In Section 2.8 we suggested a mapping between the values of the ACT–R conflict resolution parameters and two principle components of emotions correspoinding to the two–dimensional model of Russell (1983, 1989). The first dimension is *arousal* representing the strength of an emotion. The second dimension is called *valence*, and it represents whether an emotion is positive or negative. This mapping of values of arousal and valence to the goal value and relative noise was presented in Table 2.1, and it was based on the analysis of the ACT–R architecture. Now, after the results of the Dancer model have been compared with the data, we may further support the idea that some effects of emotional states on behaviour can be modelled in ACT–R using the conflict resolution parameters:

- The relation between the strength of stimulus and the goal value (G) in ACT–R models indicates that arousal can be related to G.[2]

[2]Note that constant relative noise $T = \frac{1}{G}\tau = \text{const}$ increases the noise temperature τ proportionally to

- A decrease of the relative noise (T) during successful problem solving coincides with the experience of positive emotions, such as joy, experienced by subjects on successes. Negative emotions, such as frustration on a failure during problem solving, corresponds to an increase of T in the model due to errors. Thus, the valence can be represented by changes of T.

The influence of positive and negative emotions on decision making has been noticed in subjects in a number of studies (e.g. Tversky & Kahneman, 1981; Johnson & Tversky, 1983; Nygren, Isen, Taylor, & Dulin, 1996). The mapping between some aspects of the decision framing theory and the ACT–R parameters have been also suggested in Table 2.1. In particular, it considers the effect of G and T emphasising or diminishing the influence of the expected probabilities P and costs C on decision making. In this chapter we have discussed the implications of different modes of decision making on learning and information gain in the Dancer model. It has been shown that dynamical transition between these modes that is characteristic to human and animal problem solving (noise decay and goal value increase) optimises learning and exploration processes similar to many known heuristics. Moreover, it has been shown that the dynamical control over the relative noise T not only facilitates the performance of the model, but also improves the fit with the data.

Returning to the question of what role emotion could play in learning processes, we now may speculate that by controlling the decision making strategy emotion and motivation adjust the behaviour to optimise the performance and adapt to changes in the environment.

5.7 Conclusions

In this chapter entropy reduction was used to analyse the learning in the model. The notion of entropy has been used before with regards to emotion and motivation in the series of works by Dörner and colleagues (Dörner & Hille, 1995; Bartl & Dörner, 1998). Emotion and motivation plays a great role for action control in their PSI theory. Another example of uncertainty and noise temperature used for control in cognitive architectures is the work of Hofstadter on modelling analogy (Hofstadter & Marshall, 1993; Hofstadter & Mitchell, 1994). In this chapter it was shown how to compute the entropy using internal parameters of ACT–R. This value was used to control the noise temperature in conflict resolution. This modification alone greatly improved the model fit to the data.

an increase of G.

Furthermore, the discussion in this chapter illustrates how the varying noise and goal value can help a cognitive model to learn quicker and improve its performance. A similar regulatory mechanism in the brain would allow an animal to adapt quicker to a changing environment. Moreover, it would be naive to assume that such mechanisms have not been discovered by nature during the evolution.

The adaptive changes in the body are closely related to the regulatory functions of the autonomic nervous system. We now can see that emotional and motivational mechanisms may indeed be involved also in the adaptation of the mind. Although a cognitive model cannot be considered as the proof of some processes in the brain, the results of the Dancer model suggest that the decision making and motivational mechanisms in the brain are adaptable and emotional subsystems are closely related to this adaptation.

In the next chapter the ACT–R conflict resolution theory will be revisited in an attempt to create an alternative algorithm, where the dynamics and adaptation discussed in this chapter occurs naturally. It will be shown that the new algorithm not only can reduce the number of parameters in ACT–R theory, but also has potentially wider applications for search and optimisation problems.

CHAPTER 6

Optimist: A New Conflict Resolution and Learning Algorithm

The results and analysis of the Dancer model, discussed in previous chapters, emphasised the need for a dynamic, entropy related conflict resolution mechanism. Such algorithm not only would allow cognitive models to fit better the adaptive nature of human and animal learning, but also could be potentially useful for other AI applications, because conflict resolution is in effect a search problem.

Unfortunately, ACT–R theory as it stands cannot explain the decay of noise and dynamics of motivation. In this chapter a new conflict resolution method will be described. First, the expected gain equation (2.1) will be revised. Taking from the optimistic approach of ACT–R, it will be shown that probability P_i and cost C_i can be united using the Poisson distribution. Then, a method of recursive estimation of the expected cost will be presented. A conflict between several alternative decisions will be resolved by introducing randomness into the estimated costs of the decisions. It will be shown how the method quickly converges to the most optimal solution.

It will be shown that the algorithm possesses properties of some known search and optimisation methods, such as best–first search and optimisation by simulated annealing (Kirkpatrick et al., 1983). However, the algorithm presented here is self–controlled. The method performance will be demonstrated on a search program.

Finally, the method will be compared with the ACT–R conflict resolution. The variables corresponding to the goal value and expected gain noise variance in ACT–R become dynamic: they are statistically learned during the task and environment exploration. The evolution of these variables corresponds to that suggested by cognitive models, such as the

Dancer model with the noise decay due to the entropy reduction. In addition, it will be shown that the number of parameters in ACT–R can be reduced.

6.1 Cost and Success Probability

In order to achieve the goal state a problem solver should take a series of actions that will produce changes in the system or the environment, and probably the problem solver itself (its memory, position, etc). Each action requires some effort or cost, which is associated with a loss paid by the problem solver executing the action (measured in time, energy, or monetary units). The sum of costs of all the actions required to achieve the goal will represent the cost of the whole solution path.

Many problems have multiple possible solutions. For example, there are many ways to assemble the Tower of Nottingham, and all of them may have different costs. Let us consider the conflict resolution as a search in the $x \times c$ space, where $X \in x$ are different decisions and $C \in c$ are the costs of these decisions. The number of decisions represents the *breadth* of the search, while costs define the *depth*. If costs of different solution paths leading to the goal state were known in advance, then the problem of conflict resolution could be easily solved: a solution with the smallest cost would be the favourite.

In the real world there is always a degree of uncertainty, and often even the same solution will have a slightly different cost every time it is applied to a problem. Therefore, we may consider costs as random variables.

Let $P(C)$ be a probability to solve the problem at cost C (probability that the cost is exactly C). The expected value of the cost is:

$$E\{C\} = \sum_C C\,P(C)\ ,$$

where the summation is made across all possible values of C. If cost is measured in time units, then for continuous time:

$$E\{t\} = \int_0^\infty t\,P(t)\,\mathrm{d}t\ ,$$

where $P(t)$ is the probability distribution function of times needed to achieve the goal state for $t \in [0, \infty]$. Note that the distribution function $P(t)$ defines the probability to achieve the success at any point on t. That is the value of the expected probability P for given C or G in the ACT–R utility equation (2.1).

Knowledge of the distribution functions $P(t)$ for different rules (or decisions) would allow the problem solver to calculate their expected costs $E\{t\}$, and hence would allow the choice of the best strategy. The problem is that nothing is known about the cost distributions $P(t)$. The only way to find what the distributions are is by sampling the costs of different strategies, which is by trying to solve the problem using these strategies many times.

As we can see, the whole idea of resolving the conflict using the costs seems to lose any meaning. Moreover, an example in the next section will illustrate that some costs are very hard to measure directly. It will be shown in this chapter, however, that it is not necessary to explore the solution paths in full, and that expected costs can be estimated without reaching the goal state.

6.2 Plausibility of a Solution

In order to illustrate the difficulty of sampling the costs of various strategies let us consider the following example: let the goal be to write the novel 'War and Peace' using one typewriter. Assume that we have a choice of two subjects that can type: Alexei Tolstoy and a chimpanzee (Figure 6.1). Suppose, that nothing is known about the literature talents of both typists. Moreover, let the success of the goal state only be detected when the novel is finished, and no intermediate information about the progress can be received.

It took Leo Tolstoy seven years to write 'War and Peace'. Probability theory tells us that a chimp randomly typing on the typewriter could also produce the desired result, although it would probably take several billions of years. So, theoretically using a chimpanzee can be a solution.

Of course no one would ever seriously expect a chimpanzee to write the six tomes novel. But how could a machine find the correct strategy? It may seem at first that the difference between the machine and a human choosing between such alternatives is in the knowledge available to them, and if a machine had the same amount of information, it would be able to exclude the second alternative from the consideration. But what if a human also did not have such information? In this case probably any human after waiting for some time would loose patience and try another alternative. Obviously the ability to give up on hopeless strategies without exploring them in full is an important property of human behaviour (unlike the Tower of Nottingham model with increased noise that never gave up).

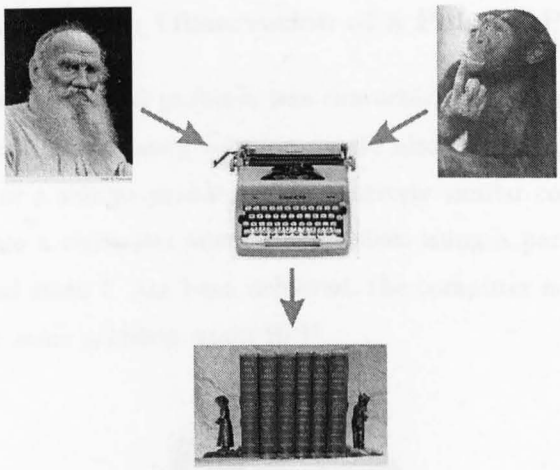

FIGURE 6.1: It took Leo Tolstoy 7 years to write 'War & Peace'. A chimp can probably type the same thing in several billions of years. How long should we wait to realise that a chimpanzee is not the best solution? The ability to give up on bad strategies in time is an important feature of human problem solving.

The challenge for a machine is to decide how soon it should give up and try another option?

What makes us think that some actions are feasible for the cause of the desired effect, but some are not, and, as a result, allows us to narrow the search? Perhaps we should provide the machine with some fitness criteria that will allow it to distinguish between good and bad solutions, which take too much effort to be found. As machines acquire more knowledge this problem will be more likely for all intelligent systems.

Let us call a feasible (practical) solution such an action (a series of actions, a choice of strategy, etc), that its expected cost $E\{C\}$ is less than some finite value G:

$$E\{C\} \leq G < \infty \, .$$

In other words, if two events, action $X \in x$ and the desired outcome $Y \in y$, are separated by a small expected cost, then we may say that X is a plausible solution for achieving goal Y. On the other hand, events separated by large expected costs are usually not considered to be related to each other. One could see problem solving as a search for these relations, that is such $x' \subseteq x$ that produce Y at small expected costs.

6.3 Problem Solving as an Observation of a Poisson Process

In previous section a solution of a problem was characterised by having a relatively small expected cost. The existence of such solution would also imply that it could be reliably repeated on the same or a similar problem with relatively similar cost.

Let us imagine a computer solving a problem using a particular algorithm, and each time after the goal state Y has been achieved, the computer is restarted and is given the task of solving the same problem again (6.2).

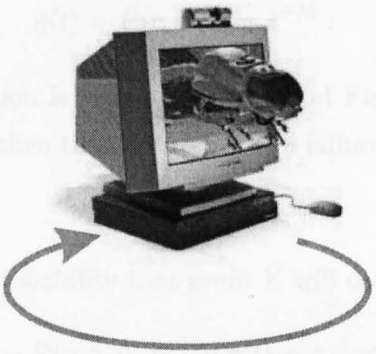

FIGURE 6.2: A computer solves the problem in a loop. If the expected cost of the algorithm is $E\{C\}$, then we shall observe the goal state at a rate $\lambda = 1/E\{C\}$.

Let the cost C of the algorithm be random with the expected value $E\{C\}$. Then we shall observe the goal state Y on average every $E\{C\}$ seconds, or at a rate $\lambda = 1/E\{C\}$. We may consider this process as an observation of the random event Y on time interval $[0, t]$ at rate λ. Such processes are known as Poisson processes, and the probability $P(n)$ of observing n events by time moment t is given by the Poisson distribution formula:

$$P(n) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} , \qquad n = 0, 1, 2, \ldots . \tag{6.1}$$

Here λ is the *mean count rate*, $n = 0, 1, 2, \ldots$ is the number of observations of the event by time moment t.

Note that when $\lambda = 0$ (or $E\{C\} = \infty$) the probability (6.1) becomes zero, which corresponds to a case when event Y is impossible. On the other hand, if the event is possible, it implies that its mean rate must be $\lambda > 0$, or $E\{C\} < \infty$. Perhaps, when solving a problem one must be *optimistic*, that is to assume that the goal state is possible:

$$\exists G < \infty : E\{C\} \leq G , \qquad (\lambda > 0) .$$

The assumption above implies that events Y are described by the Gamma distribution, and the probability of their observation is given by the Poisson formula (6.1).

6.4 Failure, Success and First Success Probabilities

Let us consider some special cases of the Poisson probability (6.1).

Failure probability is the probability that event Y will not occur ($n = 0$):

$$q(t) = P(n = 0) = e^{-\lambda t} . \tag{6.2}$$

The shape of the above function is shown on the left of Figure 6.3. One can see that if $\lambda > 0$ (goal state is possible), then the probability of a failure decreases exponentially with time.

Success probability is the probability that event Y will occur at least once ($n > 0$):

$$p(t) = P(n > 0) = 1 - q(t) = 1 - e^{-\lambda t} . \tag{6.3}$$

The success probability is shown on the right plot of Figure 6.3, one can see that it increases with time if $\lambda > 0$.

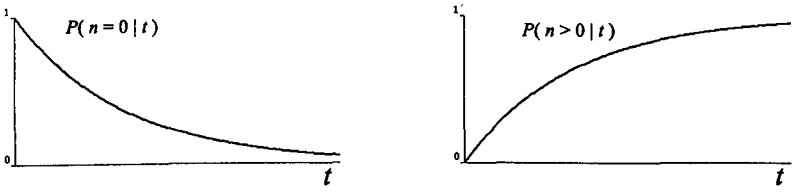

FIGURE 6.3: Left: probability of failure $P(n = 0)$ decreases with time. Right: probability of success $P(n > 0)$ increases with time.

When solving a problem, especially for the first time, what we are interested in is the first occurrence of the goal state. Moreover, often we do not need to solve exactly the same problem again. Therefore, probability of the very first success is of special interest.

First success probability is the probability that event Y will occur exactly once ($n = 1$):

$$p_1(t) = P(n = 1) = \lambda t e^{-\lambda t} . \tag{6.4}$$

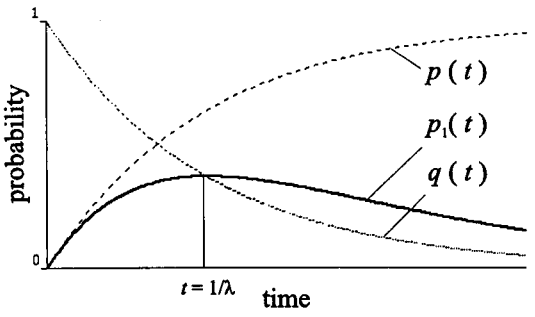

FIGURE 6.4: Probability of the first success $P(n = 1)$ has a unique maximum in $t = 1/\lambda = E\{t\}$.

The shape of the above function is shown on Figure 6.4. One may notice that it increases with time up to a certain maximum and then decreases again.

Let us find the time moment corresponding to the maximum of the first success probability:

$$\dot{p}_1(t) = \frac{\mathrm{d}}{\mathrm{d}t}\lambda t e^{-\lambda t} = 0 \quad \Rightarrow \quad t = \frac{1}{\lambda}\,.$$

We can see that this time moment corresponds to the expected cost $E\{t\}$ (most likelihood cost).

6.5 Estimation of the Expected Cost

Up to this point we have been talking about problem solving as an observation of a Poisson process with rate λ. In fact, equation (6.1) describes conditional (likelihood) probability $P(n \mid \lambda)$ of observing n events for a given value of λ (time is a parameter). In reality, when solving a problem, the rate λ is unknown, and the expected cost is what we are trying to estimate. What is known, however, is n (the number of successes) and t (the amount of time spent).

Let us estimate λ (and, hence, the expected cost $E\{C\} = \frac{1}{\lambda}$) from the posterior values of n and t. We shall consider and compare three common methods: maximum likelihood, maximum of posterior estimate, and posterior mean estimate.

Maximum likelihood Let us find the maximum of the likelihood probability (6.1) for λ:

$$\frac{\mathrm{d}}{\mathrm{d}\lambda}P(n \mid \lambda) = 0 \quad \Rightarrow \quad \lambda = \frac{n}{t} \quad \Rightarrow \quad E\{C\} \approx \frac{t}{n}\,.$$

One can use the above formula to estimate the expected cost, if the number of successes $n > 0$. Unfortunately, if no successes have been observed ($n = 0$), then the above formula cannot be used.

Maximum of posterior estimate Let us consider the posterior probability $P(\lambda \mid n)$ of λ for a given value of n and parameter t. One can show (see Appendix D) that when *a priori* all the values of λ are equally probable, and the likelihood probability $P(n \mid \lambda)$ is described by equation (6.1), then $P(\lambda \mid n)$ can be found from the likelihood probability and value of t parameter:

$$P(\lambda \mid n) = tP(n \mid \lambda) \; . \tag{6.5}$$

Note from the above formula that the probability of any $\lambda > 0$ increases with time.

Now, let us find the maximum of the posterior probability for λ:

$$\frac{\mathrm{d}}{\mathrm{d}\lambda} P(\lambda \mid n) = \frac{\mathrm{d}}{\mathrm{d}\lambda} tP(n \mid \lambda) = 0 \quad \Rightarrow \quad \lambda = \frac{n}{t} \quad \Rightarrow \quad E\{C\} \approx \frac{t}{n}$$

One can see that the above estimation is identical to the maximum likelihood, and it cannot be used if $n = 0$.

Posterior mean estimate After observing n events by time moment t we can estimate λ by calculating its expected value using the posterior probability (6.5):

$$E\{\lambda\} = \int_0^\infty \lambda P(\lambda \mid n)\, \mathrm{d}\lambda = \int_0^\infty \lambda t \, \frac{(\lambda t)^n}{n!} e^{-\lambda t}\, \mathrm{d}\lambda = \frac{n+1}{t} \quad \Rightarrow \quad E\{C\} \approx \frac{t}{n+1} \; .$$

One can see that the above formula can be used to estimate the expected cost even when no successes have yet been observed ($n = 0$). In the following sections we shall call $\bar{C} = \frac{t}{n+1}$ an estimation of the expected cost, or simply the *estimated cost.*

6.6 The Optimal Moment to Give Up

In order to optimise the process of estimation of any parameter, such as the mean count rate described in the previous section, it is important to make unbiased assessments such that no information is lost due to the way the measurements are taken. The *maximum entropy principle*, formulated by Jaynes (1957), suggests that statistical assessments should be based on probability distributions with the highest uncertainty (entropy). This ensures that the assessments of parameters are unbiased, and maximum information is extracted

from the data. Let us consider the entropy related to the probabilities of observing the goal state (entropy of success) and determine its maximum.

When observing a system solving a problem in a loop, such as shown on Figure 6.2, we may think of it as a system with random states described by n number of successes achieved by the time moment t. Let us consider the entropy of such a system:

$$H_n = -\sum_n P(n) \ln P(n) , \quad n = 0, 1, 2, ... \tag{6.6}$$

Here $P(n)$ is described by the Poisson distribution (6.1), which depends on the time parameter. Thus, the entropy H_n will also depend on different values of t. Note that the summation in (6.6) is made over all possible states $n = 0, 1, 2, ...$ However, it is not necessary to consider all these sates.

Indeed, when solving a problem we are interested in only two possible outcomes: failure ($n = 0$) and success ($n > 0$). Moreover, we may consider only the first success ($n = 1$) as one state, and a set of other outcomes as another. The entropy of such binary system can be easily calculated using equations (6.2) and (6.4) for the probabilities of these states:

$$H_{01}(t) = -q(t) \ln q(t) - p_1(t) \ln p_1(t) = \lambda t e^{-\lambda t}(1 - \ln(\lambda t e^{-\lambda t})) .$$

It is known from information theory that entropy reaches its maximum when all states of the system are equally probable. One can easily check that when $t = 1/\lambda$ the probability of a failure and the probability of the first success are equal:

$$p_1(t) = q(t) = e^{-1} , \quad t = 1/\lambda .$$

Thus, the entropy $H_{01}(t)$ reaches its maximum at $t = 1/\lambda$ (or $t = E\{t\}$). The shape of the $H_{01}(t)$ function, as well as the probabilities $q(t)$ and $p_1(t)$, are shown on Figure 6.5. Note that $t = 1/\lambda$ point also corresponds to the maximum of the first success probability (see Figure 6.4). It is also possible to find the maximum of $H_{01}(t)$ differentiating it by t.

Following the entropy maximisation principle the best moment to reassess λ is at $t = 1/\lambda$. If after the reassessment we discover that the option (rule or strategy) we are currently investigating is not the best one (i.e. there is an option with a smaller expected cost), then this also will be the moment to change the strategy. Thus, the maximum entropy principle suggests exactly how far we should explore each option, and when is the optimal moment to give up.

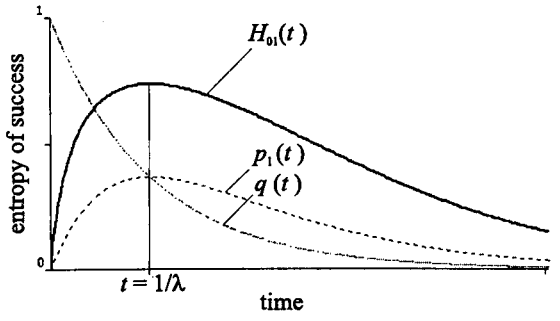

FIGURE 6.5: Entropy H_{01} of a single success as a function of time parameter t. Maximum of H_{01} corresponds to $t = 1/\lambda = E\{t\}$, where probability of failure $q(t)$ equals probability of the first success $p_1(t)$.

Interestingly, motivation theory by Atkinson (1957, 1974) (see also Revelle & Michaels, 1976) defined motivation of achievement as a function of the success and failure probabilities, such that motivation reaches its maximum when the probability of success equals the probability of failure. Thus, according to this hypothesis, which was based on a series of psychological experiments, the maximum of motivation to achieve a goal should be at $t = E\{C\} = 1/\lambda$ (see Figure 6.5), and in case of a failure the motivation to carry on, perhaps, should decline. As has been shown above, this is also the moment of maximum entropy, and it is the best moment to reassess the expected cost and possibly change strategy.

6.7 Recursive Estimation of the Expected Cost

Let us return to the example with a computer solving a problem (Figure 6.2), but with one difference: the average time (expected cost $E\{C\}$) it takes the computer to solve the problem is unknown. Also, this time we control when the computer is restarted. Our goal is to restart the computer in such a way, that the goal state appeared at the highest rate possible.

Let us denote by Δt the time intervals after which we restart the computer. If we restart the computer too late $\Delta t > E\{C\}$, then obviously the rate at which the goal state occurs will not be the highest. On the other hand, if we restart the computer too early $\Delta t < E\{C\}$, then often the computer will not have enough time to finish solving the problem.

Let us conduct a series of trials registering the first occurrence of the goal state during time intervals Δt: if the goal state is registered, then we shall restart the computer immediately after it; otherwise, we shall restart the computer after Δt. One may notice that there are only two possible outcomes of such trials (binomial or Bernoulli trials):

Failure: the goal state has not been achieved, number of successes n does not change, the overall effort (time) spent increases by $C = \Delta t$.

Success: the goal state is achieved, number of successes n increases by 1, the effort increases by $C < \Delta t$.

By counting number of successes n and summing up the time spent $t = C_1 + \cdots + C_k$ in k trials, we can estimate expected cost $E\{C\}$ using posterior mean formula:

$$E\{C\} \approx \bar{C} = \frac{t}{n+1} \ . \tag{6.7}$$

Now, starting with some small $\Delta t = C_{\min}$, let us set each next Δt equal to the last estimation of $E\{C\}$:

$$\Delta t_{k+1} = \bar{C}_k \ . \tag{6.8}$$

Note that the formula above describes reassessment of Δt at the moment of maximum entropy, because each time moment $\bar{C}_k$ has been defined by the previous estimation of the expected cost.

A typical dynamics of the estimated cost (6.7) is shown on Figure 6.6. This graph was generated by the OPTIMIST demonstration program described later in this chapter. In this example the expected cost $E\{C\}$, which was unknown to the algorithm, was set to 20, and only one single option was used in the conflict set (breadth set to 1). One can see that while $\bar{C} < 20$, no successes are registered ($n = 0$) and the estimated value $\bar{C}$ grows exponentially. When $\bar{C}$ becomes greater than 20, the system spends enough efforts (time) to register first successes ($n = 1, 2, ... > 0$). As the number of successes grows, $\bar{C}$ decreases converging to the expected cost $E\{C\} = 20$.

An example of step by step calculation of $\bar{C}_k$ and Δt_{k+1} during ten trials is shown in Table 6.1. One can show that if the expected cost is finite ($E\{C\} < \infty$), then with trials the estimated cost $\bar{C}$, and hence the restarting time Δt, will converge to $E\{C\}$:

$$\lim_{k\to\infty} \Delta t_{k+1} = \lim_{k\to\infty} \bar{C}_k = E\{C\} \ .$$

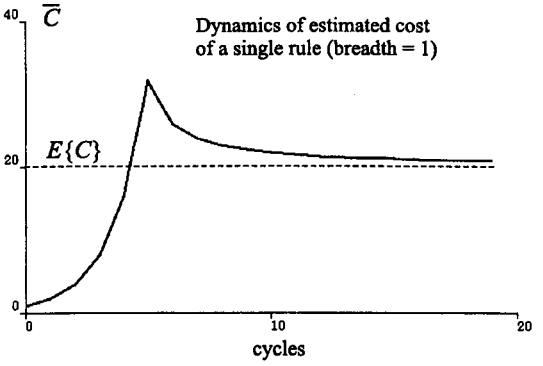

FIGURE 6.6: Estimated cost $\bar{C}$ converges to the expected cost $E\{C\}$ with cycles $k \to \infty$. The graph was generated by the OPTIMIST demonstration program with breadth set to 1 (no conflict), and the expected cost $E\{C\} = 20$ (depth).

Indeed, according d'Alembert's rule the series converges if after some kth trial the ratio:

$$\frac{\bar{C}_{k+1}}{\bar{C}_k} \leq A < 1 \; .$$

It is easy to check that on every failure (i.e. when the number of successes $n_{k+1} = n_k$) the ratio is greater than 1 and the series increases infinitely:

$$\frac{\bar{C}_{k+1}}{\bar{C}_k} = \frac{n_k + 1}{n_{k+1} + 1} \frac{C_1 + \cdots + C_{k+1}}{C_1 + \cdots + C_k} = 1 + \frac{C_{k+1}}{C_1 + \cdots + C_k} > 1 \; , \qquad (n_{k+1} = n_k) \; .$$

However, if $E\{C\} < \infty$, then at some $k+1$ step Δt_{k+1} will become greater than $E\{C\}$, and a success will be registered (i.e $n_{k+1} = n_k + 1$). The ratio then will be

$$\frac{\bar{C}_{k+1}}{\bar{C}_k} = \frac{n_k + 1}{n_{k+1} + 1} \frac{C_1 + \cdots + C_{k+1}}{C_1 + \cdots + C_k} = \frac{n_k + 1}{n_k + 2} \left(1 + \frac{C_{k+1}}{C_1 + \cdots + C_k} \right) \; , \quad (n_{k+1} = n_k + 1) \; .$$

On success the cost is less than the latest estimation of the expected cost, that is $C_{k+1} < \Delta t_{k+1} = \frac{C_1 + \cdots + C_k}{n_k + 1}$. Therefore

$$\frac{\bar{C}_{k+1}}{\bar{C}_k} = \frac{n_k + 1}{n_k + 2} \left(1 + \frac{C_{k+1}}{C_1 + \cdots + C_k} \right) < \frac{n_k + 1}{n_k + 2} \left(1 + \frac{1}{n_k + 1} \right) = \frac{n_k + 1}{n_k + 2} \frac{n_k + 2}{n_k + 1} = 1 \; .$$

Thus, the series converges if successes occur.

Because the restarting time Δt converges to $E\{C\}$, as a result, the goal state will occur at the highest rate possible. The problem of the optimal restart schedule of the so–called Las Vegas algorithms was considered by Luby, Sinclair, and Zuckerman (1993). Here we considered a similar problem using posterior estimation of the rate of a Poisson process and the maximum entropy principle.

TABLE 6.1: An example of estimation of expected cost $E\{C\}$ in 10 trials. A failure occurred in the first two ($k = 1, 2$, $n = 0$) and successes occurred in the following eight trials. Here k is the trial number, n is the number of successes, $\bar{C}_k$ is the estimated cost, and Δt_{k+1} is the maximum waiting time on the next trial.

k	n	$\bar{C}_k$	Δt_{k+1}
0	0	$C_{\min}$	$\bar{C}_0$
1	0	Δt_1	$\bar{C}_1$
2	0	$\Delta t_1 + \Delta t_2$	$\bar{C}_2$
3	1	$\frac{\Delta t_1+\Delta t_2+\Delta t_3}{2}$	$\bar{C}_3$
...	...	...	...
10	8	$\frac{\Delta t_1+\cdots+\Delta t_{10}}{9}$	$\bar{C}_{10}$

6.8 Resolving the Conflict

In previous sections we discussed how to estimate the cost of one particular algorithm (decision or production rule) by estimating the rate of a hypothetical Poisson process. As an illustration we used the example of a computer set into an endless loop performing the algorithm. Now, returning to the problem of conflict resolution, let us consider a choice of several computers tackling the same problem (Figure 6.7). However, only one computer can be used at a time. Let us denote the alternatives (computers) by x, and suppose each of them uses different algorithm with a different expected cost $E\{C(x)\}$. Our goal is to find the fastest (cheapest) x.

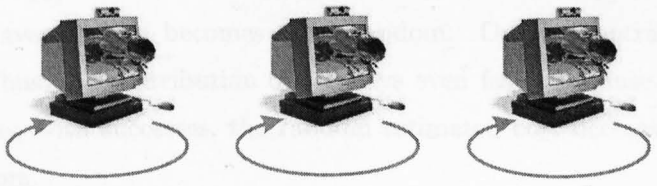

FIGURE 6.7: The conflict represented by several computers using different algorithms to solve the problem. The goal is to find the one with the fastest solution.

Let us record for each option x the following information: $k(x)$ — the number of times x was used, $n(x)$ — number of successes for x, $t(x) = C_1(x) + \cdots + C_{k(x)}(x)$ — the efforts (all time) spent using x. Now, after trying each option we can estimate its expected cost:

$$E\{C(x)\} \approx \bar{C}(x) = \frac{t(x)}{n(x)+1} .$$

In order to resolve the conflict we introduce a *random estimated cost*:

$$\tilde{C}(x) = \frac{k(x)\,\bar{C}(x) + \xi(\bar{C}(x))}{k(x)+1}\ , \tag{6.9}$$

where $\xi(\bar{C}(x))$ is a random variable, such that its expected value is equal to the estimated cost $\bar{C}(x)$:

$$E\left\{\xi\left(\bar{C}(x)\right)\right\} = \bar{C}(x)\ .$$

We shall call $\xi(\bar{C}(x))$ a *random prediction*. For example, we can use the following function: $\xi(\bar{C}(x)) = \text{rand} \in (0, 2\bar{C}(x))$. One can see that expected value of the random estimated cost (6.9) is also equal to the estimated cost:

$$E\left\{\tilde{C}(x)\right\} = \bar{C}(x)\ .$$

By looking at equation (6.9) one can see that the random estimated cost consists of two components: one based on the estimated cost $\frac{k\bar{C}}{k+1}$, and another one is the random predication $\frac{\xi}{k+1}$. If $k(x) = 0$ (option x has not yet been used), then $\tilde{C}(x)$ is determined entirely by the random $\xi(C_{\min})$, where $C_{\min}$ is the initial smallest cost. As the number of trials increases ($k(x) > 0$), the contribution of ξ becomes less noticeable.

However, if the number of failures increases ($n(x) \ll k(x)$), then $\bar{C}(x)$ grows exponentially, and much faster than $k(x)+1$ in the divisor of equation (6.9). As a result, not only the expected value of $\tilde{C}(x)$ grows exponentially, but also the contribution of ξ increases, because $E\{\xi\} = \bar{C}$ by definition. This way, with failures, the random estimated cost increases on average and becomes more random. On the contrary, with successes $\bar{C}(x)$ decreases. Thus, the contribution of ξ decays even faster because its expected value collapses. This way, with successes, the random estimated cost decreases on average and becomes less random.

The conflict between several alternative x is resolved by selecting the one with the smallest random estimated cost:

$$x = \arg\min\left[\tilde{C}(x)\right]\ .$$

Thus, the algorithm is performing cost optimisation. As in previous sections, the maximal waiting time for the next binomial trial is determined by the estimated cost of the last chosen x:

$$\Delta t_{j+1} = \bar{C}_j(x)\ .$$

Note that here j is the cycle number and it is not the same as $k(x)$ — the number of times x has been chosen. This is because on different cycles j different options x can be used, and $j = \sum_x k(x)$.

Because $E\{\tilde{C}(x)\} = \bar{C}(x)$ by definition, the random estimated cost $\tilde{C}(x)$ has the same limit as $\bar{C}(x)$ (i.e. $\tilde{C}(x)$ converges if $E\{C(x)\} < \infty$). Thus, if finite solutions exist, then the algorithm will eventually find the one with the smallest expected cost (i.e. the optimal solution).

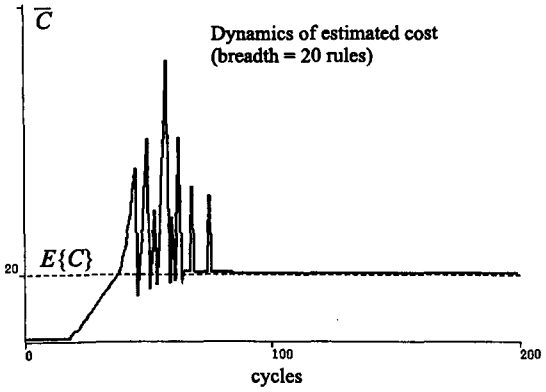

FIGURE 6.8: An example of the dynamics of the estimated cost $\bar{C}(x)$ converging to the optimal value. The plot was generated by the OPTIMIST demonstration program with breadth set to 20.

Figure 6.8 shows an example of the dynamics of the estimated cost during 200 cycles. This graph was generated by the OPTIMIST program (described later in this chapter) with the conflict set of 20 alternatives (breadth set to 20). As in the example shown on Figure 6.6, the expected cost of the optimal solution was set to 20. One can see that prediction $\bar{C}$ converges to the optimal expected cost $E\{C\} = 20$.

6.9 Method Performance

In this section a program demonstrating some properties of the OPTIMIST conflict resolution method will be described and its performance discussed.

6.9.1 OPTIMIST Demonstration Program

A program demonstrating the OPTIMIST method has been implemented in Common Lisp and GARNET. The interface of the program is shown on Figure 6.9, which presents the

search space with alternatives x represented on the horizontal axis, and costs c on the vertical axis. A set of gadgets controls the size of the search space (breadth and depth). Figure 6.9 shows breadth set to 20 (i.e. a conflict set of 20 alternatives).

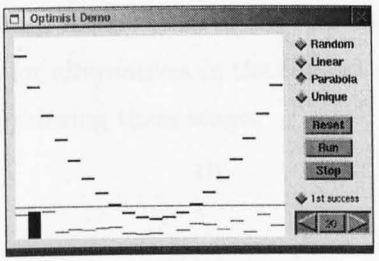

FIGURE 6.9: Interface of the OPTIMIST demonstration program

When one of the alternatives is selected by the algorithm, it is shown by a vertical beam rising up from the corresponding x position (e.g. the second entry is selected on Figure 6.9). The height of the beam represents the depth of the search, that is the maximal cost the program is currently paying investigating the selected alternative (Δt). Thus, the higher the values of estimated costs $\bar{C}$, the deeper the search Δt, and the taller the beams. Horizontal line on Figure 6.9 represents the current Δt defined after the outcome of the last trial (success or failure).

Successes or failures of the selected alternatives are determined by the interaction with the environment (outside world), which is represented in the program by a distribution of the real costs. These are the costs, which are not known to the problem solver initially (i.e. expected costs $E\{C(x)\}$). The goal is to find an alternative with the optimal cost is located (i.e. x with the smallest $E\{C(x)\}$). The real costs are represented on the screen by thick horizontal bars. Their distribution shape is controlled by a user. Figure 6.9 shows parabolic distribution with the smallest (optimal) cost positioned in the middle. Other distributions available are random, linear, and unique distribution.

If the cost paid by the algorithm on selected x is enough to achieve the goal (the beam rises higher than the real cost bar on that position), then a success is registered. Otherwise, a failure occurs. Posterior information ($k(x)$, $n(x)$, and $t(x)$) is used to estimate $\bar{C}(x)$. Estimated costs for all entries x are stored in the memory of the program and are represented by thin horizontal bars.

6.9.2 Program Performance

The program demonstrates the behaviour of the algorithm with four distinguishable stages. These stages are shown on four screenshots of Figure 6.10 taken during a run of the program with 50 alternatives and parabolic distribution of the real costs. Figure 6.11 illustrates the dynamics of choice proportion for alternatives in the conflict set, and Figure 6.12 illustrates the dynamics of Δt (and $\bar{C}(x)$) during these stages.

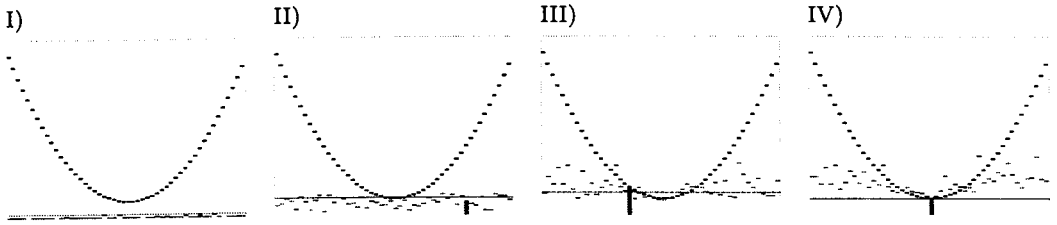

FIGURE 6.10: Search for the optimal solution in parabolic distribution with 50 alternatives.

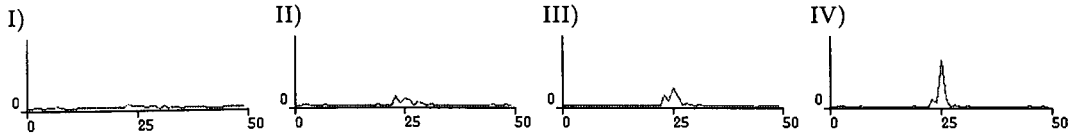

FIGURE 6.11: Dynamics of choice proportion of alternatives in the conflict set.

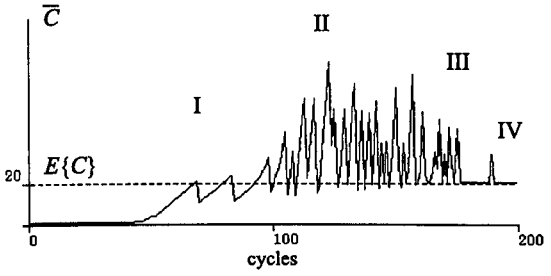

FIGURE 6.12: Dynamics of the estimated cost (optimal cost set to 20) for a conflict set of 50 alternatives (breadth 50). Four stages compared with thermodynamic system: I — heating up, II — boiling, III — cooling down, IV — crystallisation.

In the beginning the search is completely random and not very deep, so many alternatives are being selected, and the heights of the beams are small (Figure 6.10, I). The choice of alternatives in the beginning is very random and broad (Figure 6.11, I). However,

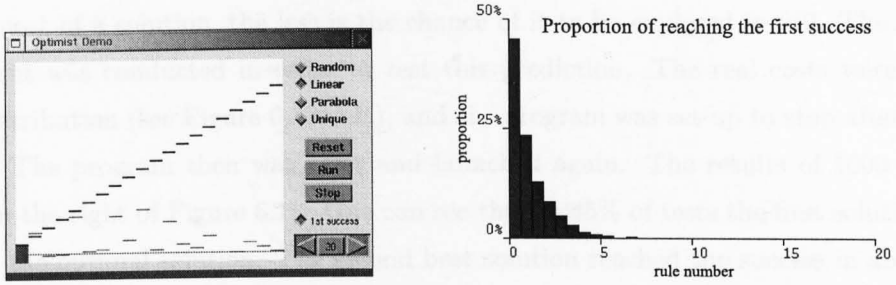

FIGURE 6.13: Left: OPTIMIST simulation with real costs set to a linear distribution, and stop after the first success option switched on. The first rule from left is the optimal solution, and it is shown to have reached the success. Right: proportion in per cent showing how often each rule becomes the first to reach the success. In 452 tests out of 1000 (45%) the first solution found was also the optimal one, and in another 23% of cases it was the second best solution.

when no successes are registered the estimated costs begin to grow exponentially increasing the search depth (stage I on Figure 6.12).

When the depth explored by the algorithm becomes greater than some real costs, the first successes start to occur. For some period of time the number of successes is comparable with the number of failures (Figures 6.10 and 6.11, stage II). On successes Δt decreases, and increases on failures (see area II on Figure 6.12).

When the the more optimal solutions are found, the system starts to concentrate on the better entries (stage III on Figures 6.10 and 6.11). Because the system registers more successes than failures, Δt decreases on average (area III on Figure 6.12).

Finally, the system stabilises choosing only the optimal alternative (Figures 6.10 and 6.11, stage IV) and exploring the depth just enough to reach the success (area IV on Figure 6.12).

Following the analogy with thermodynamic systems, these four stages can be compared with heating up (I), boiling (II), cooling down (III), and crystallisation (IV) processes.

The algorithm is highly adaptive: if the distribution of the real costs changes, then the system becomes random again because Δt increases on failures. This process continues until the algorithm finds another solution. Each time the environment changes, however, it takes the system more time to adapt.

The first solution the algorithm finds is not necessarily the optimal. However, it can be shown that the first is more likely to be the optimal solution. Indeed, the greater is

the real cost of a solution, the less is the chance of it to be explored in full. The following experiment was conducted in order to test this prediction. The real costs were set to a linear distribution (see Figure 6.13, left), and the program was set up to stop after the first success. The program then was reset and launched again. The results of 1000 tests are shown on the right of Figure 6.13. One can see that in 45% of tests the first solution found was also the optimal solution. The second best solution reached the success in about 23%, and third best was lucky only in 12% of tests. Thus, in almost 70% of cases the first solution found was either the best or second best solution. The results of such experiments, however, depend greatly on the distribution of the real costs: if a solution is unique, then the first found will also always be the optimal (because there are no other solutions); if there are many similar solutions, then any of them can be found first with equal probability.

6.9.3 Search Heuristics and Optimisation

The method implements the best–first search strategy making a smooth transition from a breadth–first to a depth–first search (see Figures 6.10 and 6.11). It is possible to notice similarities of the behaviour of the method with the optimisation by simulated annealing (Kirkpatrick et al., 1983). The distinguishable feature of the OPTIMIST algorithm is that here the so–called annealing schedule (increase or decrease of the temperature of the system) is controlled automatically. Moreover, the maximum entropy principle suggests that the resulting schedule is close to the optimal.

6.10 Analogies with Neural Network Theories

The behaviour of the random estimated cost function (6.9) recalls of two important properties known from neural networks theories: plasticity and reinforcement learning. It was suggested by Sejnowski (1977a, 1977b) that neural plasticity can be explained by the so–called *covariance learning* rule, which is a variation on the Hebb rule. According to this rule, the firing rate of incoming signal (presynaptic) is compared with the neuron's own average firing rate (average postsynaptic), resulting in decrease or increase of the synapse. It was proposed also by Bienenstock, Cooper, and Munro (1982) that the postsynaptic information about the average of the neuron's own firing rate is stored in the threshold of a neuron, and it changes over time. However, with time the period over which the postsynaptic activity is averaged become longer, and, as a result, it takes longer to change the neuron's own

threshold and adapt. On the contrary, because the average own rate of a young neuron is not settled enough, the behaviour is more random and synapses are more plastic. This effect can explain why synaptic plasticity of some neurons decreases with age.

One can see that random estimated cost functions (6.9) manifest similar plasticity properties: the larger the number k of times x has been used, the larger the number of samples over which the random estimated cost is calculated, and consequently the more samples are needed to change its expected value.

It is necessary to point out that the method requires an interaction with the environment (or the problem space), which should provide the answer (success or failure) to the system. In fact, such interaction between the components of a feedback system has been viewed as defining the behaviour of animals (Baum, 1973) and later propagated on the level of individual neurons (Sutton & Barto, 1981).

The information acquired during the binomial tests of rules acts similarly to reward or penalty signals in reinforcement learning theories (Barto, 1985; Barto & Anandan, 1985). Indeed, initially all alternatives x have equal chances to be selected from the conflict set, because no posterior information is available ($t(x) = 0$, $k(x) = 0$, $n(x) = 0$), and the choice is based entirely on random predictions ξ. As was shown earlier, with failures the value of $\bar{C}(x)$, and consequently the expected value of $\tilde{C}(x)$, increases. As a result, the chance of the failed rule to be selected on the next trial decreases. This is similar to the inhibition effect in neural networks. On the contrary, when successes occur the value of $\bar{C}(x)$ and the expected value of $\tilde{C}(x)$ decrease. This leads to excitation of the successful alternative x, because its chance to be selected next time becomes higher.

Note that the excitatory or inhibitory information acquired with a success or failure does not act as just binary reward or penalty signals. In addition to binary change of the number of successes $n(x)$, there is also continuous C_k added to the efforts sum $t = C_1 + \cdots + C_k$ changing the estimated cost $\bar{C}$. The significance of the change in $\bar{C}$ value depends on the distance between the previous estimation and realisation: $\bar{C} - C$. If the realisation C is much smaller than the estimated value ($C \ll \bar{C}$), then the reward information will be more significant. So, there may be rewards of different degrees. Similarly, negative distance $\bar{C} - C$ will result in different degrees of penalty.

One may notice that if we represent the cost only by the time spent executing actions, and we stop testing an alternative at $\Delta t = \bar{C}$, then the realisation C cannot be greater than $\bar{C}$. Therefore, on failures the distance will always be zero, and hence penalty

will always have the same value. However, if we consider other objectives (e.g. the amount of energy spent per time, sense of comfort or danger), then we shall be able to see realisations greater than the predicted values. This way the penalties can also have different values.

6.11 Comparison with the ACT–R parameters

Let us take a look again at the ACT–R conflict resolution and its parameters and compare them with the method proposed here. The first parameter in the utility equation (2.1) is expected probability P calculated empirically by equation (2.5). In the OPTIMIST notation this empirical probability would correspond to $\frac{n+1}{k+1}$. Although probability is not used in the OPTIMIST algorithm, one can use equations (6.2), (6.3) and (6.4) to calculate the values of failure, success and the first success probabilities for any Δt.

The goal value parameter (G) in ACT–R specifies the maximal cost at which the problem solver is expected to achieve the goal. In the OPTIMIST conflict resolution method it corresponds to Δt, which is defined by the most recent $\bar{C}$ (see equation (6.8)). The value of $\bar{C}$, however, may change during problem solving (see Figures 6.6, 6.8, and 6.12), whereas G in ACT–R is constant.

The next parameter in utility equation (2.1) is cost C, which is defined in ACT–R by equation (2.6) as the average effort. In OPTIMIST notation it would correspond to $\frac{t}{k+1}$. It is, however, not used by the OPTIMIST method.

Finally, expected gain noise $\xi(s)$ in ACT–R corresponds to the random prediction $\xi(\bar{C})$. The difference is that in ACT–R noise variance is constant, whereas in OPTIMIST the expected value of $\xi(\bar{C})$ is determined by the estimated cost ($\bar{C}$), which increases on failures and decreases on successes. Moreover, the contribution of ξ depends also on $i(x)$ creating the effect of noise decay with experience.

Note that $k(x)$ is specific for every rule x, thus making noise in random estimated costs of all rules different for every rule, and it depends on how often a rule has been used in the past. This property realises Taatgen's idea to replace the production strength parameter in ACT–R.

Table 6.2 summarises the parameters and equations used in both methods. Note that although there is the same statistical information collected in both methods (n, k and t), there are fewer parameters in the OPTIMIST method. Another important difference is that the two static parameters, the goal value G and noise temperature τ, correspond to one

TABLE 6.2: Comparison of parameters and their calculations between the ACT–R conflict resolution mechanism and OPTIMIST method.

ACT–R		OPTIMIST	
number of successes	$n+1$	number of successes	n
successes + failures	$k+1$	number of trials	k
actual efforts spent	t	actual efforts spent	t
expected probability	$P=\frac{n+1}{k+1}$	see equations (6.2), (6.3), and (6.4)	
average cost	$C=\frac{t}{k+1}$		
goal value	G	estimated cost	$\bar{C}=\frac{t}{n+1}$
noise	$\xi(\tau)$	random prediction	$\xi(\bar{C})$
noise temperature	τ	estimated cost	$\bar{C}=\frac{t}{n+1}$
expected gain E	$PG-C+\xi$	random estimated cost	$\tilde{C}=\frac{k\bar{C}+\xi(\bar{C})}{k+1}$
conflict resolution	$\max E$	conflict resolution	$\min \tilde{C}$

dynamic variable $\bar{C}$. The dynamics of noise and estimated cost in OPTIMIST corresponds to the dynamics of G and τ suggested by several cognitive models (see Chapter 5).

6.12 Optimist Conflict Resolution for ACT–R

In order to test the OPTIMIST conflict resolution on a working model, an overlay replacing the standard conflict resolution of ACT–R has been written. It uses several hook functions of ACT–R and allows switching between the two conflict resolution mechanisms. The code of the OPTIMIST overlay is presented in Appendix C. It must be said that this is a very early implementation of the overlay and so far it has been tested only with ACT–R Version 4 running the Dancer model. However, the first results, which will be presented here, are quite encouraging.

6.12.1 Conflict Resolution by Random Estimated Costs

The first encouraging results are the fact that the model does run with using the new algorithm and is able to accomplish the task. The model learns new rules and which of the rules are better. The left plot of Figure 6.14 illustrates a typical trace of the number errors produced by the model. The plot on the right shows the dynamics of ACT–R probabilities for productions involved in choosing the door. One can see that the number of productions increases.

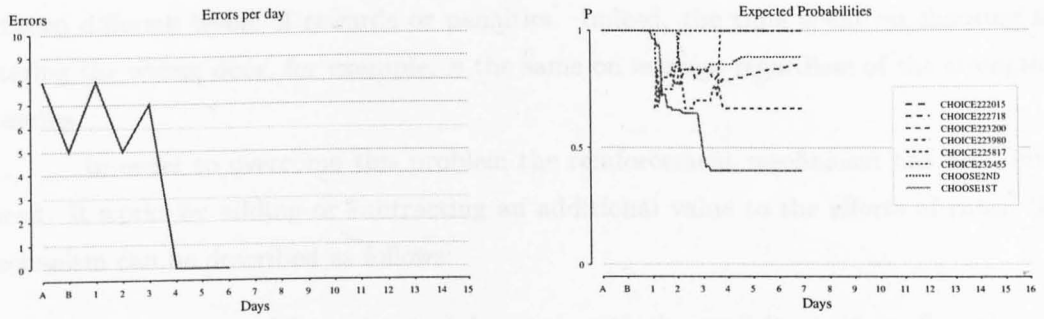

FIGURE 6.14: Left: Errors trace produced by the model running with the OPTIMIST conflict resolution. Right: Dynamics of standard ACT–R expected probabilities P_i for rules involved in choosing the door.

These probabilities, however, are no longer used by the conflict resolution. Instead, the parameters of each rule (see Table 6.2) are used to compute their expected costs $\bar{C}$ and the conflict is resolved by choosing the smallest randomised estimated cost $\tilde{C}$. The dynamics of $\bar{C}$s is shown on the left plot of Figure 6.15. For comparison the standard ACT–R costs of these rules are shown on the right plot. One may notice that although the costs of some rules are small (e.g. for rules *choose1st* and *choose2nd*) their expected costs may be quite large.

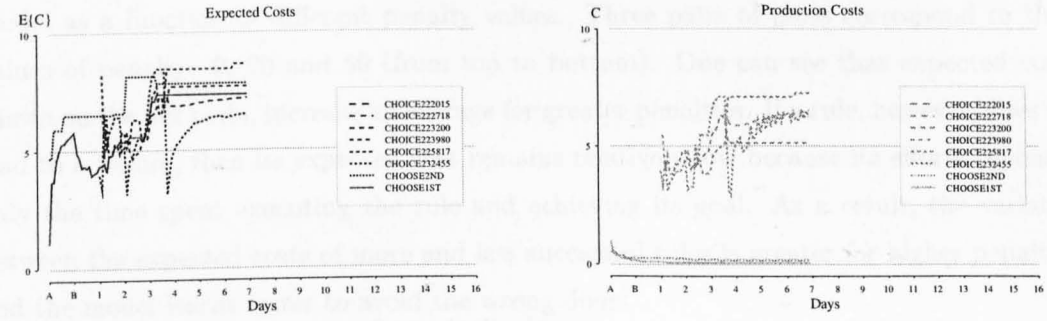

FIGURE 6.15: Left: Dynamics of expected costs $\bar{C}_i$ for rules involved in choosing the door. Right: Dynamics of standard ACT–R costs C_i for rules involved in choosing the door.

6.12.2 Reinforcement

Because OPTIMIST no longer uses the goal value parameter (G), it was necessary to come up with a solution to present different levels of stimulation. Indeed, if the cost is only measured by the time spent on a rule and its goal, then there is no way to distinguish

between different levels of rewards or penalties. Indeed, the time spent on choosing and entering the wrong door, for example, is the same on average regardless of the strength of stimulus.

In order to overcome this problem the reinforcement mechanism has been introduced. It works by adding or subtracting an additional value to the efforts of rules. The mechanism can be described as follows:

- If a goal is removed from the stack by a rule with the explicit `:failure` flag, then the *penalty* value is added to the efforts of a rule that set the goal.
- If a goal is removed from the stack by a rule with the `:success` flag, then the *reward* is subtracted from the efforts of a rule that set the goal.
- The values of penalty and reward are defined by the corresponding variables, which can be controlled from the simulation (see Appendix C).

This mechanism is possibly not the best way to implement reinforcement, but it allowed the Dancer model to simulate the behaviour for different values of strength of stimulus. The implementation of the reinforcement mechanism can be reviewed in the future.

Figure 6.16 shows the dynamics of expected costs and the performance of the model as a function of different penalty values. Three pairs of plots correspond to three values of penalty: 0, 20 and 50 (from top to bottom). One can see that expected costs, shown on the left plots, increase on average for greater penalties. If a rule, however, does not lead to a failure, then its expected cost remains relatively low because its efforts represent only the time spent executing the rule and achieving its goal. As a result, the variation between the expected costs of more and less successful rules is greater for higher penalties, and the model learns faster to avoid the wrong door.

6.12.3 Global Noise

The model with the OPTIMIST conflict resolution demonstrated unusually fast learning: once the correct rules have been compiled, the number of errors decayed much quicker than in the data. This can be explained by the following reasons:

- Real mice could consider more features of the doors (e.g. smell) and as a result could learn more rules to choose the door.

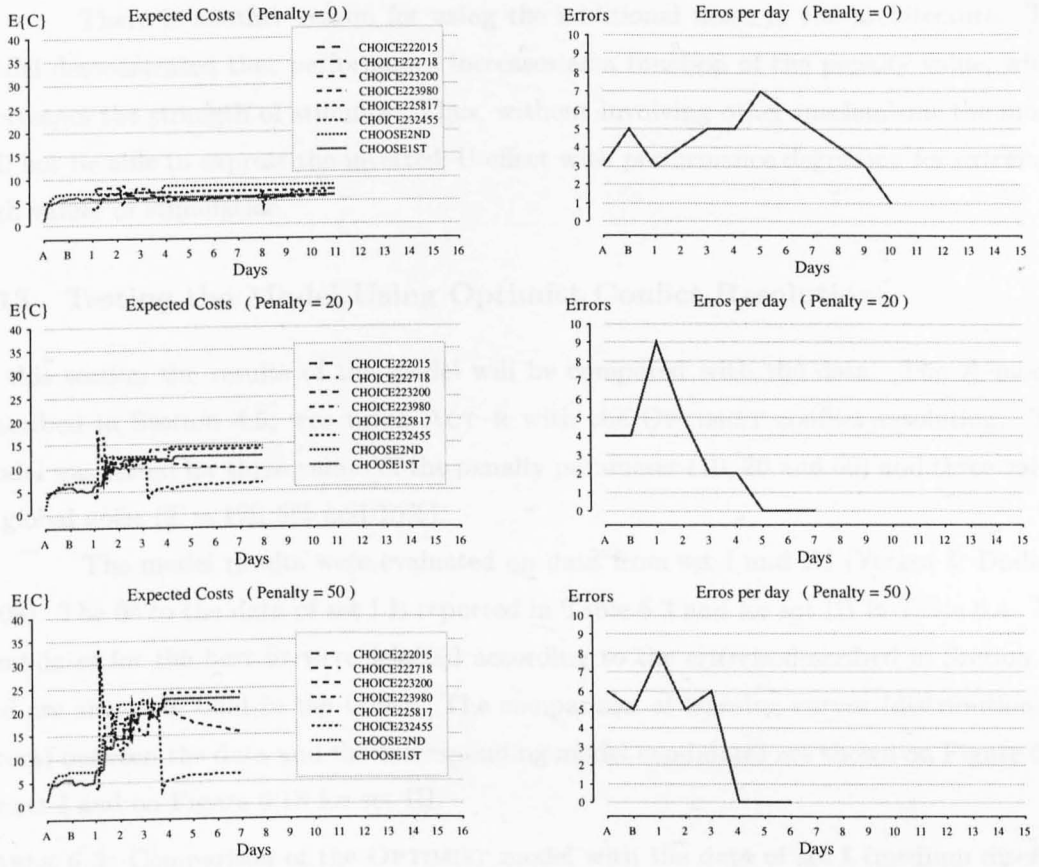

FIGURE 6.16: Model performance as a function of penalty value: Penalty 0 (Top), 20 (Middle) and 50 (Bottom). Left: Dynamics of expected costs $\bar{C}_i$; Right: Errors produced by the model.

- The model and the mechanism is too idealistic.

In order to be able to adjust slightly the performance of the algorithm the OPTIMIST overlay provides a mechanism to add noise to already randomised expected costs:

$$\tilde{C}(x) = \frac{k\,\bar{C}(x) + \xi(\bar{C}(x))}{k(x)+1} + \zeta(T \cdot \bar{C}(x)) \;, \tag{6.10}$$

where ζ is the *global expected cost noise* — a random value with zero mean and variance determined by $T \cdot \bar{C}(x)$. Here T sets the variance as the ratio of expected cost $\bar{C}$. Using different values of T it is possible to make the behaviour of the OPTIMIST conflict resolution less rational.

There is another reason for using the additional noise in the architecture. The model demonstrated that performance increases as a function of the penalty value, which represents the strength of stimulus. Thus, without involving other mechanisms the model will not be able to express the inverted–U effect with performance degrading for extremely high values of stimulation.

6.13 Testing the Model Using Optimist Conlict Resolution

In this section the results of the model will be compared with the data. The *A*–model, described in Section 4.6, was using ACT–R with the OPTIMIST conflict resolution. The model was tested for three values of the penalty parameter (10, 20 and 50) and three values of global noise ($T = 1\%$, 5% and 15%).

The model results were evaluated on data from set I and III (Yerkes & Dodson, 1908). The fit to the data of set I is reported in Table 6.3 and for set III in Table 6.4. The candidates for the best fit were selected according to the criteria described in Section 4.2 and are shown in bold in the tables. The comparison of learning curves (distributions of errors) between the data and the corresponding model candidates are shown on Figure 6.17 for set I and on Figure 6.18 for set III.

TABLE 6.3: Comparison of the OPTIMIST model with the data of set I (medium discrimination). Data for different stimulation levels compared with model performance under different penalty values and minimal noise (T). The best matches in terms described in Section 4.2 are shown in bold and on Figure 6.17.

Stimulation		125		300		500	
T	Penalty	rms	R^2	rms	R^2	rms	R^2
1%	10	**14.9%**	**.51**	17.6%	.80	15.1%	.74
	20	15.2%	.61	**17.7%**	**.83**	14.9%	.84
	50	18.0%	.58	23.7%	.78	19.3%	.77
5%	10	14.4%	.40	21.0%	.68	19.2%	.77
	20	12.1%	.62	18.0%	.83	14.5%	.82
	50	14.4%	.67	16.3%	.77	12.1%	.90
10%	10	16.8%	.53	26.1%	.61	25.4%	.67
	20	11.5%	.55	19.5%	.63	15.0%	.84
	50	12.6%	.68	18.4%	.71	**12.3%**	**.89**

One can see that the results of the model with the OPTIMIST conflict resolution are slightly better than those of the model with the traditional mechanism (see Tables 4.3, 4.4 and Figures 4.14, 4.15 for comparison). In particular, the errors decay faster and the

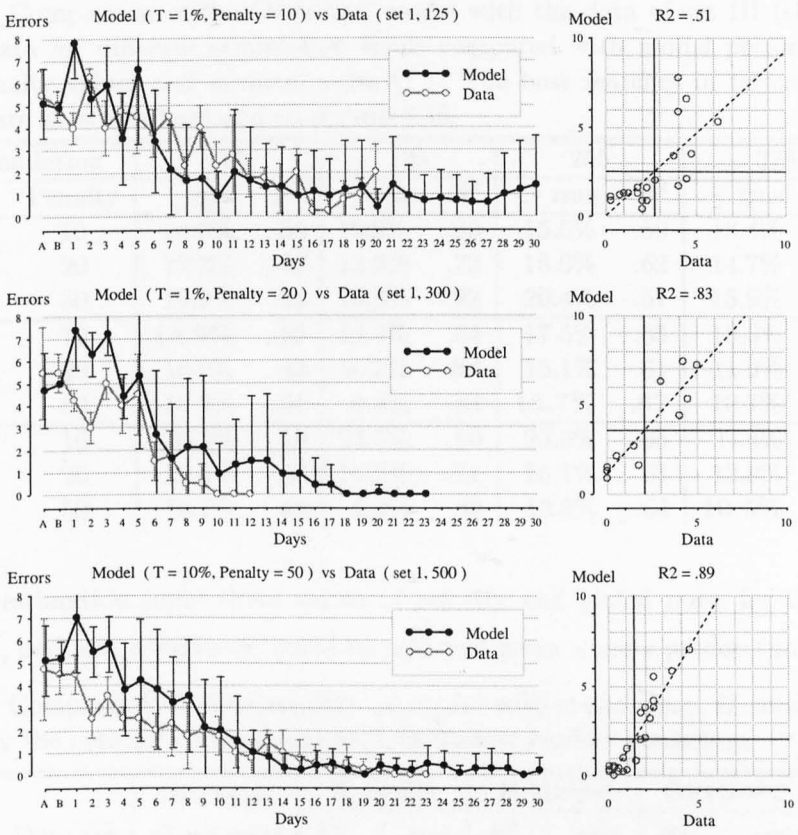

FIGURE 6.17: Comparison of the results of model with OPTIMIST conflict resolution with the data set I (medium visual discrimination). Left: learning curves. Right: regression plots.

curves correspond better in the final stages of the experiments. Note that in order to achieve this effect with the standard ACT–R mechanism it was necessary to gradually reduce the expected gain noise variance during the model run by controlling it artificially through the entropy parameter. As predicted, the new mechanism expresses a more adaptive behaviour without the involvement of other computations (e.g. entropy). Indeed, the learning curves and the fit of the OPTIMIST model are more similar to those produced by the H–model in Section 5.2.

Table 6.5 shows the summary comparing the results of the three models: A–model with standard mechanism, H–model with entropy controlled noise and the OPTIMIST model. One may see that the H–model produced the best fit, and the OPTIMIST model the second best. It is necessary, however, to point out that these results are based on a very imprecise

TABLE 6.4: Comparison of the OPTIMIST model with the data of set III (slight discrimination). Data for different stimulation levels compared with model performance under different penalty values and minimal noise (T). The best matches in terms described in Section 4.2 are shown in bold and on Figure 6.18.

Stimulation		135		195		255		375	
T	Penalty	rms	R^2	rms	R^2	rms	R^2	rms	R^2
1%	10	18.0%	.34	10.9%	.80	15.5%	.59	13.4%	.61
	20	19.8%	.43	12.9%	.73	15.9%	.62	14.7%	.60
	50	19.0%	.54	10.3%	.92	20.4%	.57	15.9%	.70
5%	10	**14.9%**	**.40**	18.3%	.64	17.5%	.65	18.0%	.56
	20	16.0%	.43	**9.2%**	**.87**	15.1%	.61	11.9%	.71
	50	18.9%	.51	9.3%	.84	**13.7%**	**.67**	10.7%	.77
10%	10	20.2%	.19	22.7%	.56	23.0%	.65	24.4%	.41
	20	12.9%	.53	14.8%	.71	15.1%	.61	12.9%	.70
	50	16.7%	.54	9.9%	.82	13.8%	.64	**10.5%**	**.75**

parameters estimation (only three values of penalty and global noise for the OPTIMIST model), and, perhaps, a better fit could be achieved given a more precise tuning.

TABLE 6.5: Comparison of three models: A–model with static noise, H–model with noise controlled by the entropy and a model with OPTIMIST conflict resolution.

	Strength	A–model		H–model		OPTIMIST	
Data set	of stimulus	R^2	rms	R^2	rms	R^2	rms
Set I	125	.54	12.2%	.64	10.1%	.51	14.9%
	300	.77	13.2%	.86	8.8%	.83	17.7%
	500	.82	12.4%	.88	7.1%	.89	12.3%
Set III	135	.73	11.4%	.50	14.5%	.40	14.9%
	195	.82	10.2%	.90	9.4%	.87	9.2%
	255	.69	14.5%	.71	8.4%	.67	13.7%
	375	.61	12.5%	.68	9.5%	.75	10.5%

Although the OPTIMIST model demonstrated some promising results, it does not fully explain the nature of the inverted–U effect. Indeed, as noted earlier, the performance of the model increases as a function of penalty (strength of stimulus), and it was necessary to use an increased global noise T parameter in order to simulate the effect of an extremely strong stimulus and to obtain a U–shaped performance curve (see Figure 6.19). A higher initial value of relative noise T_0 was also required by the H–model to match the data for a strong stimulus.

Explaining the inverted–U effect was not the priority of this thesis. However, let

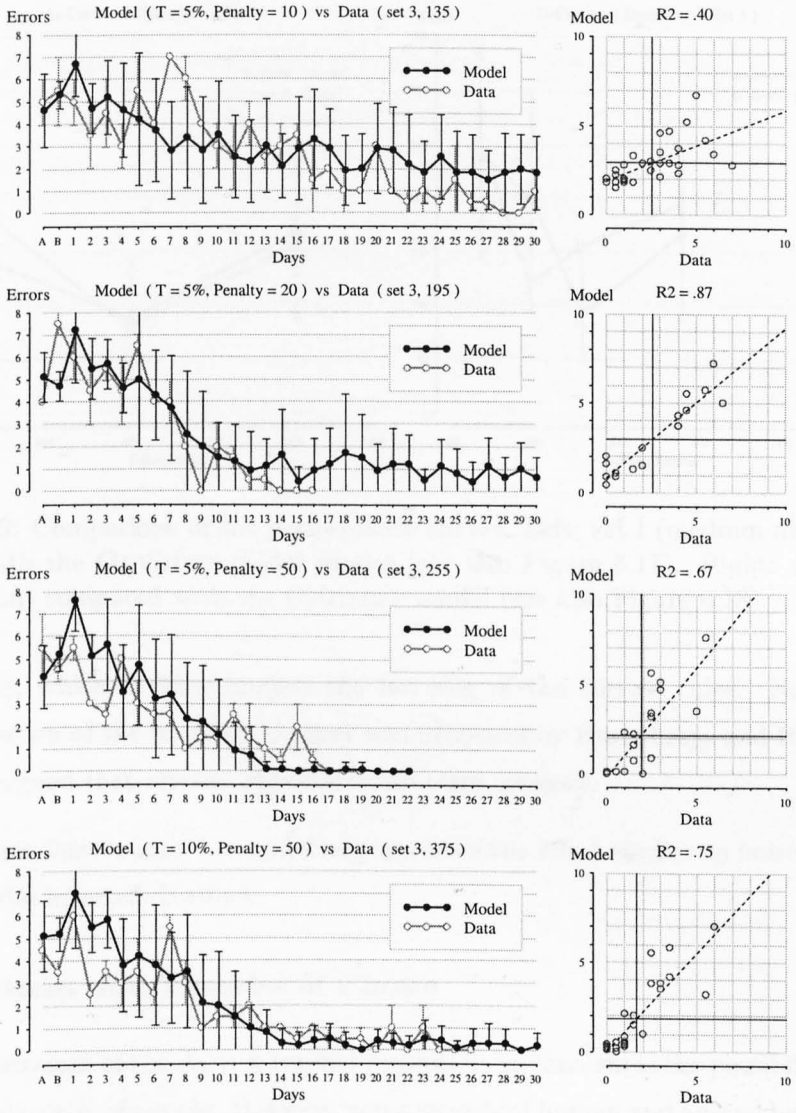

FIGURE 6.18: Comparison of the results of model with OPTIMIST conflict resolution with the data from set III (slight visual discrimination). Left: learning curves. Right: regression plots.

us identify possible reasons for the degradation of performance:

- There are some additional rules learned under a strong stimulus (e.g. a rule to avoid the choice altogether), which can make it more difficult to choose the correct rule.
- Strong stimulus affects the use of declarative knowledge units (e.g. the colour features

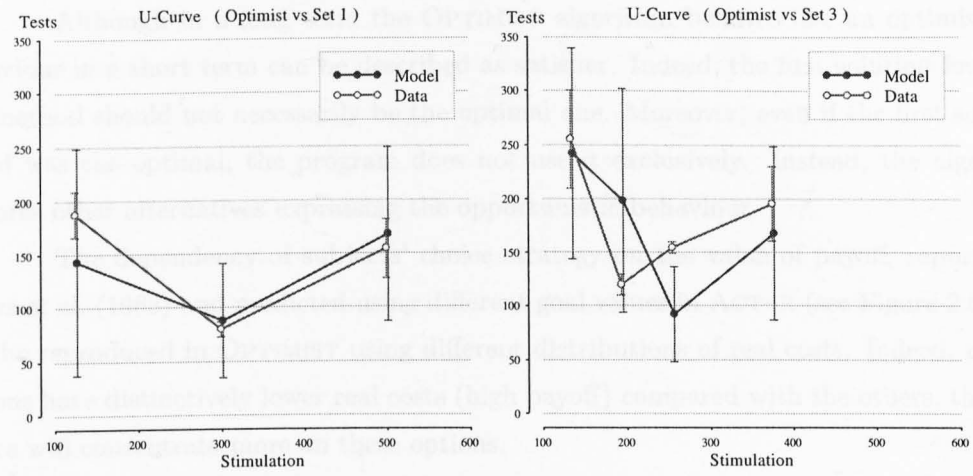

FIGURE 6.19: Comparison of the performance curves. Left: set I (medium discrimination) compared with the OPTIMIST model results (see also Figure 6.17). Right: set III (slight discrimination) compared with the OPTIMIST model (see also Figure 6.18).

chunks), which in turn hinders the learning of the correct rules. Note that such explanation of the inverted–U effect was proposed by Humphreys and Revelle (1984). They argued that arousal impedes short–term memory.

Including these factors into the model may produce the effect similar to noise increase and account for the inverted–U effect.

6.14 Optimist and Theories of Choice

One of the successes of the ACT–R conflict resolution mechanism is the possibility to predict the choice behaviour of people. However, some models of human and animal behaviour, such as the Tower of Nottingham model (Jones et al., 2000) and the Dancer model described in this study, indicated the need for the revision of the ACT–R mechanism to incorporate more dynamics. The new algorithm proposed here not only implements the additional properties, but also possesses properties of the current method.

One may argue that humans are satisfiers rather than optimisers, and that the solution we usually use is not necessarily the optimal. Moreover, even if people have tried the optimal solution once, they still may try other alternatives on some occasions. This opportunistic behaviour of subjects was reported by Friedman et al. (1964), and it was reproduced in ACT–R models by Anderson (1990, 1993), Anderson et al. (1993).

Although in a long term the OPTIMIST algorithm behaves like an optimiser, its behaviour in a short term can be described as satisfier. Indeed, the first solution found by the method should not necessarily be the optimal one. Moreover, even if the first solution found was the optimal, the program does not use it exclusively. Instead, the algorithm explores other alternatives expressing the opportunistic behaviour.

The dependency of subjects' choice strategy on the value of payoff, reported by Myers et al. (1963) and predicted using different goal values in ACT–R (see Figure 2.3), can also be reproduced in OPTIMIST using different distributions of real costs. Indeed, if some options have distinctively lower real costs (high payoff) compared with the others, then the choice will concentrate more on these options.

Finally, the OPTIMIST algorithm finds support in several studies on kinetics of choice. First, a theory of the effect of reinforcement proposed by Baum (1973) suggested the correlation law between the response rate and the rate of reinforcement (see also Herrnstein, 1961). Later a kinetic interpretation of this law was derived by Myerson and Miezin (1980), and it used the Poisson distribution to explain the response frequency. This hypothesis was supported by experimental data in a study of the response rates of rats (Mark & Gallistel, 1994), which also suggested that more recent rewards are more important.

These studies have not remained unnoticed by the ACT–R community, and the events discounting mechanism, mentioned in Section 2.3, was introduced in ACT–R to improve the conflict resolution and parameters learning mechanisms (Lovett & Anderson, 1996; Lovett, 1998). The new conflict resolution algorithm suggested in this study directly uses the posterior Poisson distribution to estimate the rate of success (i.e. the rate of reinforcement). Thus, it provides a better support for the theories on kinetics of choice and reinforcement.

6.15 Summary

In this chapter conflict resolution was considered as a search and optimisation problem. The cost of a solution was considered as a random variable, and its posterior mean estimate was derived from Poisson distribution. Optimal schedule for the restarts strategy was suggested using the maximum entropy principle, and the method of choosing the alternatives by comparing their random estimated costs was presented.

The performance of the method has been demonstrated on random distributions,

and properties of the method were compared with other heuristic techniques. It was shown that the method implements the dynamics and adaptive properties suggested by the cognitive models as well as some theories of animal and human learning.

The Dancer model using the new conflict resolution method has been tested and compared with the data. The working model proves that the OPTIMIST conflict resolution method can be used as an alternative to theory of choice behaviour. The model using the OPTIMIST algorithm demonstrated a more adaptable behaviour than with the standard ACT–R conflict resolution.

CHAPTER 7

Discussion

The contributions and implications of this work span over three areas: improving cognitive models, understanding of emotion within intelligent systems and creating new theory and algorithm for use in other areas of AI and computer science. The plans for the future work will be outlined in the end of this chapter.

7.1 Contributions of This Work to Cognitive Modelling

In the introduction chapter of this thesis the need to include the effects of emotion into cognitive models was argued. The case was made that many theories and experiments suggest the important role that emotion plays in intelligence. However, in cognitive modelling even the most famous effects, such as the inverted–U law relating arousal (one of the principle components of emotions) with cognitive performance, have been ignored so far. Although several theories in psychology have tried to explain the inverted–U phenomenon by proposing various effects of arousal on cognitive processes (e.g. Easterbrook, 1959; Humphreys & Revelle, 1984) these theories have not been tested on a model. This work has presented for the first time an ACT–R model of the well–known work in the area — the Yerkes and Dodson experiment.

The effects of emotion and motivation on judgement have been discussed before by Tversky and Kahneman (1981) in their decision framing theory (see also Tversky & Kahneman, 1974; Johnson & Tversky, 1983). This theory have been compared with the decision making mechanism of ACT–R (conflict resolution). The influence of the conflict resolution parameters (goal value and noise variance) on performance of the architecture (and models) has been analysed. Based on this analysis, the mapping between the conflict

resolution parameters and the principle components of emotions, arousal and valence, has been suggested. In addition, the effect of production strength learning and activations of working memory elements (chunks) has been considered.

The model has demonstrated how these parameters can be used to simulate the inverted–U effect. In particular, modifications of goal value and expected gain noise variance proved to be very effective. This supports the idea that emotion is greatly involved in human and animal decision making. In addition, the idea that arousal impedes short–term memory (Humphreys & Revelle, 1984), which may explain the inverted–U effect, has been considered and supported.

It has been shown that although different settings of the ACT–R conflict resolution parameters can be used to represent different modes of affective decision making (see Table 2.1), the current theory does not explain the dynamics of transition between these modes. The lack of such dynamics in ACT–R has been associated with problems in some other cognitive models, as mentioned in Section 2.5. In Chapter 5 of this thesis a simple method was proposed to incorporate the decay of noise by controlling it with the entropy parameter. This modification made the decision making of ACT–R more dynamic and adaptable with the noise variance decaying with experience. Moreover, the noise may increase if an unexpected error occurs. Such an increase of noise on failure may result in the model choosing more diverse strategies, and may represent changes in behaviour associated with emotional experiences during problem solving (e.g. joy or distress changing the arousal and valence). The greater diversity of strategies used by subjects than by the associated cognitive models was attributed to emotion (Dörner, 2001). The Dancer model using dynamic noise demonstrated a better fit with the data than models with static noise. Thus, incorporating the entropy controlled noise can improve the performance of these models.

Computations of entropy have been used for control in cognitive architectures before (e.g. Dörner & Hille, 1995; Hofstadter & Marshall, 1993). This work demonstrated how entropy can be calculated using parameters of the ACT–R architecture and fed back to control the model. In addition, this work presented entropy reduction as a tool for assessing the quality of learning in cognitive models under different parameter settings. Thus, the work showed how information theory combined with cognitive modelling can provide a new framework for investigating the effects of various factors on the quality (or rapidity) of learning.

Although the entropy control can provide a temporary solution for making noise

more dynamic, the current conflict resolution method does not explain the dynamics of another parameter — the goal value. As was mentioned in Section 5.4, the achievement motivation theories (Atkinson, 1957, 1974; Revelle & Michaels, 1976) further suggest the idea that goal value parameter of ACT–R should not remain static. In Chapter 6 of this thesis the current ACT–R conflict resolution theory was revised, and a new method proposed. The new method, called OPTIMIST, incorporates naturally the dynamic transition from a random low effort search (high noise, low goal value) to a deterministic highly motivated decision making behaviour (low noise, high goal value). The main insight of the method — to match and maximise the rate of an unknown Poisson process — is not entirely new and was proposed to explain the choice behaviour of animals (Baum, 1973; Myerson & Miezin, 1980; Mark & Gallistel, 1994). The thesis provided an algorithm that allows us to incorporate such behaviour in cognitive models.

An implementation of the OPTIMIST algorithm as an overlay for ACT–R has been provided. The overlay connects easily to existing models and provides the mechanism to switch between the standard and the new conflict resolution methods without modifying the model itself. The overlay was tested on the Dancer model and the results are promising: similarly to the model with dynamic noise the match between the model and data has improved.

The thesis does not imply that all cognitive models ought to take emotions into account. However, based on the results of this work, we may now summarise those types of models that could benefit from considering different arousal and motivational states:

- Models of tasks where many different strategies can be used, and where these strategies are represented by several conflicting productions. If these models learn the probabilities and costs of these productions, then the entropy controlled noise (5.3) can improve the model performance.

- Models where the activations of chunks and errors of omission as a result of retrieval failure influence the performance should consider that chunks' base level activation may differ with respect to the level of arousal.

7.2 On a Role of Emotion in Learning

The results of the model described in this thesis as well as the results of some other models indicate that a model with dynamic control of the conflict resolution parameters improved the fit of the model to data. The changes of these parameters could reflect the shifts in judgements and decision strategies of subjects as they progress in learning about the task. These changes coincide with intermediate failures and successes during problem solving, which are as we know accompanied by emotions such as frustration or joy.

This research showed how arousal affects decision making in a particular model, and also the implications of such changes on problem solving and learning process. Using entropy reduction as a tool to assess the information gained by the system, it was shown that there are situations where increased amounts of noise facilitates the rate of information acquisition. In particular, an increase of noise at the beginning of a new problem, on environmental changes, and on failures helps the system to learn more quickly. Furthermore, an increase of motivation from low to high during problem solving optimises the efforts spent on searching for the solution.

This work showed the similarities between the suggested dynamic control over the conflict resolution parameters and some search and optimisation methods already known and used elsewhere in AI. Thus, if indeed the changes in decision making of subjects are the result of emotions and autonomic arousal, then we speculate that the interaction between ANS and cognition are an important heuristic function that improves the efficiency of learning processes. The lack of such interaction could explain the oddly unintelligent behaviour of the patients described by Damasio (1994) (see discussion in Section 1.1).

The function of emotion and its positive contribution to intelligence has been the subject of many studies in philosophy, psychology and artificial intelligence. This thesis has presented a different approach within the framework of a unified theory of cognition.

7.3 New Methods in Computer Science

Computer science has seen many examples recently of how discoveries in biology, psychology and neuroscience quickly find applications in data processing, optimisation and computation. Methods such as genetic algorithms, ant colony optimisation, competitive learning and self–organising maps are just a few examples.

In this work using a modern cognitive modelling technique we considered an important phenomenon of human and animals psychology — emotion and particularly its role in decision making and learning. Some experiments on decision making suggest that our judgements do not always agree with sophisticated statistical inferences. When making decisions, people often rely on some simple, flawed, but mostly efficient heuristics (Tversky & Kahneman, 1974). As noted by Tversky and Kahneman, decision making systems could be improved if they took into account the predictive nature of human mind, whose focus is often on 'What will I feel then?' rather than on 'What do I want now?' (Tversky & Kahneman, 1981, page 458).

Inspired by the results of the model the current conflict resolution method of ACT–R has been revised, and a new algorithm proposed. The new algorithm has similar properties to the current method, and yet it delivers new dynamics which is known to be consistent with those that are characteristic for human and animal behaviour. The method makes few assumptions about the distribution of solutions and, in fact, it is only assumed that solutions exist (optimism principle). It has been shown using the maximum entropy principle how deep each alternative should be explored and what is the optimal moment to give up and try another option. As has been demonstrated on a simple search program, the method quickly finds a good and in many cases the optimal solution. There are similarities with the existing optimisation and search methods, such as simulated annealing, best–first search, tabu search and maximum gradient methods. However, the complete autonomic and self–adaptive behaviour of the method is its particular advantage. Such methods are especially useful in situations where not much is known about the distribution of the solutions. These properties suggest that the algorithm can be used as a meta–heuristic for many problems outside the cognitive modelling domain, such as machine learning, robotics, decision making, search and optimisation.

The conflict resolution method presented in this thesis aims to create an alternative theory of decision making to the methods currently used in the ACT–R architecture. However, the demonstrated properties and cheap computation cost of the algorithm may prove to be useful for solving other problems in computer science. Although the method is yet to be tested on some classical optimisation problems with large search spaces, the introduction of a new technique is a valuable addition to the arsenal of the existing methods in AI.

7.4 Future Work

There are several projects currently considered as the directions for the future work. The following four developments derive from the research presented in this thesis:

1. Further testing and improvement of the OPTIMIST overlay for the ACT–R architecture. The new conflict resolution should be tested on models of many experiments on probability matching and choice behaviour (e.g. Friedman et al., 1964; Myers et al., 1963; Herrnstein, 1961). The results should be compared with the existing models (Lovett & Anderson, 1995, 1996).

2. Application of the OPTIMIST algorithm to classical optimisation problems such as the Travelling Salesman problem. The comparison of the method with the existing techniques will help to determine the weaknesses of the method, and possibly its advantages.

3. It would also be interesting to transfer the Dancer model into ACT–R Version 5 and to use the perceptual–motor capabilities (ACT–RPM, Byrne and Anderson (1998)). The asynchronous behaviour of ACT–R 5 is potentially more useful for the new conflict resolution scheme.

7.5 Summary

The work considers the problem of a function of emotion in cognition by modelling the important effect of arousal on cognitive performance. The phenomenon known as the inverted–U effect has been explained by several theories, which have however not been tested on a computational model. The results of the work favour the idea that emotion is important for intelligence. Combining information theory and cognitive modelling the work demonstrated explicitly on a model how changes in decision making strategy improve learning similar to some known heuristic methods. The thesis suggested methods of incorporating the effects of arousal on performance in cognitive models. The conflict resolution mechanism of ACT–R has been revisited and a new theory has been proposed. The new algorithm, based on estimating the rate of a Poisson process, adds a more dynamic approach to the conflict resolution and makes the behaviour more adaptable. In addition, the new decision making method can find application in other areas of AI and computer science.

References

Anderson, J. R. (1990). *The adaptive character of thought.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J. R. (1993). *Rules of the mind.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J. R., Kushmerick, N., & Lebiere, C. (1993). The Tower of Hanoi and goal structures. In J. R. Anderson (Ed.), *Rules of the mind* (pp. 121–142). Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought.* Mahwah, NJ: Lawrence Erlbaum Associates.

Anderson, J. R., Lebiere, C., & Lovett, M. (1998). Performance. In J. R. Anderson & C. Lebiere (Eds.), *The atomic components of thought* (pp. 57–100). Mahwah, NJ: Lawrence Erlbaum Associates.

Anderson, K. J. (1990). Arousal and the inverted–U hypothesis: A critique of Neiss's "Reconceptualizing arousal". *Psychological Bulletin, 107*(1), 96–100.

Anderson, K. J., & Revelle, W. (1982). Impulsivity, caffeine, and proof–reading: A test of the Easterbrook hypothesis. *Journal of Experimental Psychology: Human Perception and Performance, 8*, 614–624.

Atkinson, J. W. (1957). Motivational determinants of risk–taking behavior. *Psychological Review, 64*, 359–372.

Atkinson, J. W. (1974). Strength of motivation and efficiency of performance. In J. W. Atkinson & J. O. Raynor (Eds.), *Motivation and achievement.* Washington D.C.: V. H. Winston.

Bard, P. (1934). Emotion: I. The neuro–humoral basis of emotional reactions. In C. Murchison (Ed.), *A handbook of general experimental psychology* (pp. 264–311). Worcester: Clark University Press.

Bartl, C., & Dörner, D. (1998). PSI: A theory of the integration of cognition, emotion and motivation. In F. E. Ritter & R. M. Young (Eds.), *Proceedings of the Second European Conference on Cognitive Modeling* (pp. 66–73). Nottingham, UK: Nottingham University Press.

Barto, A. G. (1985). Learning by statistical cooperation of self–interested neuron–like computing elements. *Human Neurology, 4*, 229–256.

Barto, A. G., & Anandan, P. (1985). Pattern–recognizing stochastic learning automata. In *IEEE Transactions on Systems, Man and Cybernetics* (Vol. 15, pp. 360–375).

Bates, J., Loyall, A. B., & Reilly, W. S. (1992, May). *An architecture for action, emotion, and social behaviour.* Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Baum, W. M. (1973). The correlation–based law of effect. *Journal of the Experimental Analysis of Behavior, 20*, 137–153.

Baxter, G. D., & Ritter, F. E. (1996). *Designing abstract visual perceptual and motor action capabilities for use by cognitive models* (Tech. Rep. No. 36). ESRC CREDIT, Psychology, University of Nottingham.

Belavkin, R. V. (2001). The role of emotion in problem solving. In C. Johnson (Ed.), *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing* (pp. 49–57). Heslington, York, England: AISB.

Belavkin, R. V., Ritter, F. E., & Elliman, D. G. (1999). Towards including simple emotions in a cognitive architecture in order to fit children's behaviour better. In *Proceedings of the 1999 Conference of the Cognitive Science Society* (p. 784). Mahwah, NJ: Lawrence Erlbaum.

Bienenstock, E. L., Cooper, L. N., & Munro, P. W. (1982). Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience, 2*, 32–48.

Broen, W. E. J., & Storms, L. H. (1961). A reaction potential ceiling and response. *Psychological Review, 68*, 405–415.

Brown, R., & Kulik, J. (1977). Flashbulb memories. *Cognition, 5*, 73–99.

Byrne, M. D., & Anderson, J. R. (1998). Perception and action. In *The atomic components of thought* (pp. 167–200). Mahwah, NJ: Lawrence Erlbaum Associates.

Cahn, J. E. (1990). The generation of affect in synthesized speech. *Journal of the American Voice I/O Society, 8*, 1–19.

Cannon, W. B. (1915, 1929). *Bodily changes in pain, hunger, fear and rage.* New York: Appleton.

Damasio, A. R. (1994). *Descartes' error: Emotion, reason, and the human brain.* New York, NY: Gosset/Putnam Press.

Dörner, D. (2001). Strategies in a complex game and their background. In E. M. Altmann, A. Cleeremans, C. D. Schunn, & W. D. Gray (Eds.), *Proceedings of the Fourth International Conference on Cognitive Modeling* (pp. 241–242). Mahwah, NJ: Lawrence Erlbaum.

Dörner, D., & Hille, K. (1995). Artificial souls: Motivated emotional robots. In *IEEE Conference Proceedings, International Conference on Systens, Man, and Cybernetics; Intelligent Systems for 21st Century* (Vol. 4 to 5, pp. 3828–3932). Vancouver.

Easterbrook, J. A. (1959). The effect of emotion on cue utilization and the organization of behaviour. *Psychological Review, 66*, 183–201.

Erman, L. D., Hayes-Roth, F., Lesser, V., & Reddy, D. (1980). The HARSLEY II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys, 12*(2), 213–253.

Fleetwood, M. D., & Byrne, M. D. (2001). Modeling icon search in ACTR–R/PM. In E. M. Altmann, A. Cleeremans, C. D. Schunn, & W. D. Gray (Eds.), *Proceedings of the Fourth International Conference on Cognitive Modeling* (pp. 17–22). Mahwah, NJ: Lawrence Erlbaum.

Friedman, M. P., Burke, C. J., Cole, M., Keller, L., Millward, R. B., & Estes, W. K. (1964). Two–choice behaviour under extended training with shifting probabilities of reinforcement. In R. C. Atkinson (Ed.), *Studies in mathematical psychology* (pp. 250–316). Stanford, CA: Stanford University Press.

Frijda, N. H. (1986). *The emotions.* Cambridge University Press.

Frijda, N. H., & Swagerman, J. (1987). Can computers feel? Theory and design of an emotional system. *Cognition and Emotion, 1*(3), 235–257.

Gobet, F., & Jansen, P. (1994). Towards a chess program based on a model of human memory. In H. J. van der Herik, I. S. Herschberg, & J. W. H. M. Uiterwijk (Eds.), *Advances in computer chess* (Vol. 7, pp. 35–60). Maastricht, The Netherlands: University of Limburg.

Goleman, D. (1995). *Emotional intelligence.* New York: Bantam Books.

Grant, D. A. (1962). Testing the null hypothesis and the strategy and tactics of investigating theoretical models. *Psychological Review, 69*(1), 54–61.

Gupta, B. S. (1977). Dextroamphetamine and measures of intelligence. *Intelligence, 1*, 274–280.

Hartline, H. K., Wagner, H. G., & Ratcliff, F. (1956). Inhibition in the eye of limulus. *Journal of General Physiology, 39*(5), 651–673.

Hebb, D. O. (1955). Drives and the C.N.S. (conceptual nervous system). *Psychological Review, 62*, 243–254.

Herrnstein, R. J. (1961). Relative and absolute strength of response as a function of frequency of reinforcement. *Journal of the Experimental Analysis of Behavior, 4*, 267–272.

Hinton, G. E., Sejnowski, T. J., & Ackley, D. H. (1984). *Boltzmann machines: Constraint satisfaction networks that learn* (Tech. Rep. No. 119). Carnegie–Mellon University.

Hofstadter, D. R., & Marshall, J. B. D. (1993). *A self–watching cognitive architecture of high–level perception and analogy–making* (Tech. Rep. No. 100). Indiana University Center for Research on Concepts and Cognition.

Hofstadter, D. R., & Mitchell, M. (1994). The Copycat project: A model of mental fluidity and analogy–making. In K. Holyoak & J. Barnden (Eds.), *Advances in connectionist and neural computation theory, volume 2: Analogical connections* (pp. 31–112). Ablex.

Hudlicka, E., & Fellous, J.-M. (1996). *Review of computational models of emotion* (Tech. Rep. No. 9612). Arlington, MA: Psychometrix.

Humphreys, M. S., & Revelle, W. (1984). Personality, motivation, and performance: A theory of the relationship between individual differences and information processing. *Psychological Review, 91*(2), 153–184.

James, W. (1884). What is an emotion? *Mind, 9*, 188–205.

Jaynes, E. T. (1957). Information theory and statistical mechanics. *Physical Review, 106, 108*, 620–630, 171–190.

Johnson, E., & Tversky, A. (1983). Affect, generalization, and the perception of risk. *Journal of Personality and Social Psychology, 45*, 20–31.

Jones, G., Ritter, F. E., & Wood, D. J. (2000). Using a cognitive architecture to examine what develops. *Psychological Science, 11*(2), 93–100.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, J. M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Lambie, J. A., & Marcel, A. J. (2002). Consciousness and the varieties of emotion experience: A theoretical framework. *Psychological Review, 109*(2), 219–259.

Lebiere, C., & Anderson, J. R. (1993). A connectionist implementation of the ACT–R production system. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society* (pp. 635–640). Hillsdale, NJ: Lawrence Erlbaum Associates.

Lebiere, C., & Anderson, J. R. (1998). Cognitive arithmetic. In *The atomic components of thought* (pp. 297–342). Mahwah, NJ: Lawrence Erlbaum Associates.

LeDoux, J. E. (1990). Information flow from sensation to emotion: Plasticity in the neural computation of stimulus value. In M. Gabriel & J. Moore (Eds.), *Learning and computational neuroscience: Foundations of adaptive networks* (pp. 3–51). MIT Press.

LeDoux, J. E. (1996). *The emotional brain.* New York: Simon & Schuster.

Liebert, R. M., & Morris, L. W. (1967). Cognitive and emotional components of test anxiety: A distinction and some initial data. *Psychological Reports, 20,* 975–978.

Lovett, M. C. (1998). Choice. In *The atomic components of thought* (pp. 255–296). Mahwah, NJ: Lawrence Erlbaum Associates.

Lovett, M. C., & Anderson, J. A. (1995). Making heads or tails out of selecting problem solving strategies. In *Proceedings of the 17th Annual Conference of the Cognitive Science Society* (pp. 265–270). Hillsdale, NJ: Lawrence Erlbaum Associates.

Lovett, M. C., & Anderson, J. A. (1996). History of success and current context in problem solving: Combined influences on operator selection. *Cognitive Psychology, 31,* 168–217.

Lovett, M. C., Daily, L. Z., & Reder, L. M. (2000). A source activation theory of working memory: Cross–task prediction of performance in ACT–R. *Journal of Cognitive Systems Research, 1,* 99–118.

Luby, M., Sinclair, A., & Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. In *Israel Symposium on Theory of Computing Systems* (pp. 128–133).

Mandler, G., & Sarason, S. B. (1952). A study of anxiety and learning. *Journal of Abnormal and Social Psychology, 47,* 166–173.

Mark, T. A., & Gallistel, C. R. (1994). Kinetics of matching. *Journal of Experimental Psychology, 20*(1), 79–95.

Matthews, G. (1985). The effects of extraversion and arousal on intelligence test performance. *British Journal of Psychology, 76,* 479–493.

Myers, B. A., Guise, D. A., Dannenberg, R. B., Zanden, V. V., Kosbie, D. S., Pervin, E., Mickish, A., & Marchal, P. (1990). Garnet: Comprehensive support for graphical, highly–interactive user interfaces. *IEEE Computer, 23*(11), 71–85.

Myers, J. L., Fort, J. G., Katz, L., & Suydam, M. M. (1963). Differential monetary gains and losses and event probability in a two–choice situation. *Journal of Experimental Psychology, 77,* 453–359.

Myerson, J., & Miezin, F. M. (1980). The kinetics of choice: An operant systems analysis. *Psychological Review, 87*(2), 160–174.

Näätänen, R. (1973). The inverted–U relationship between activation and performance: A critical review. In S. Kornblum (Ed.), *Attention and performance iv* (pp. 155–174). New York: Academic Press.

Neiss, R. (1990). Ending arousal's reign of error: A reply to Anderson. *Psychological Bulletin, 107*(1), 101–105.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, Massachusetts: Harvard University Press.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice–Hall.

Nygren, T. E., Isen, A. M., Taylor, P. J., & Dulin, J. (1996). The influence of positive affect on the decision rule in risk situations. *Organizational Behavior and Human Decision Processes, 66*, 59–72.

Oatley, K., & Johnson-Laird, P. N. (1987). Towards a cognitive theory of emotions. *Cognition and Emotion, 1*(1), 29–50.

Ortony, A., Clore, G. L., & Collins, A. (1988). *The cognitive structure of emotions.* Cambridge, MA: Cambridge University Press.

Picard, R. W. (1997). *Affective computing.* Cambridge, Massachusetts. London, England: MIT Press.

Plutchik, R. (1994). *The psychology and biology of emotion* (1st ed.). New York: HarperCollins College Publishers.

Post, E. L. (1943). Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics, 65.*

Revelle, W., & Michaels, E. J. (1976). The theory of achievement motivation revisited: The implications of inertial tendencies. *Psychological Review, 83*(5), 394–404.

Ritter, F. E. (1993). Three types of emotions that can occur in a cognitive architecture like Soar. In *Workshop on Architectures Underlying Motivation and Emotion.*

Ritter, F. E., Baxter, G. D., Jones, G., & Young, R. M. (2000). Supporting cognitive models as users. *ACM Transactions on Computer–Human Interaction, 7*(2), 141–173.

Roseman, I. J., Antoniou, A. A., & Jose, P. E. (1996). Appraisal determinants of emotions: Constructing a more accurate and comprehensive theory. *Cognition and Emotion, 10*(3), 241–277.

Rosenbloom, P., & Newell, A. (1987). Learning by chunking: A production system model of practice. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development* (pp. 221–288). Cambridge, MA: MIT Press.

Russell, J. A. (1983). Two pan–cultural dimensions of emotion words. *Journal of Personality and Social Psychology, 45*, 1281–1288.

Russell, J. A. (1989). Measures of emotion. In *The measurement of emotions* (Vol. 4). New York: Academic Press.

Salovey, P., & Mayer, J. D. (1990). Emotional intelligence. *Cognition and Personality, 9*(3), 185–211.

Salvucci, D. D., & Macuga, K. L. (2001). Predicting the effect of cell–phone dialing on driver performance. In E. M. Altmann, A. Cleeremans, C. D. Schunn, & W. D. Gray (Eds.), *Proceedings of the 2001 Fourth International Conference on Cognitive Modeling* (pp. 25–30). Mahwah, NJ: Lawrence Erlbaum.

Schachter, S., & Singer, J. E. (1962). Cognitive, social, and psychological determinants of emotional state. *Psychological Review, 69*, 379–399.

Scherer, K. R. (1993). Studying the emotion–antecedent appraisal process: An expert system approach. *Cognition and Emotion, 7*, 325–355.

Scheutz, M., & Logan, B. (2001). Affective vs. deliberative agent control. In C. Johnson (Ed.), *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing* (pp. 1–10). Heslington, York, England: AISB.

Schoppek, W., Holt, R. W., Diez, M. S., & Boehm-Davis, D. A. (2001). Modeling behavior in complex and dynamic situations — the example of flying an automated aircraft. In E. M. Altmann, A. Cleeremans, C. D. Schunn, & W. D. Gray (Eds.), *Proceedings of*

the Fourth International Conference on Cognitive Modeling (pp. 265–266). Mahwah, NJ: Lawrence Erlbaum.

Sejnowski, T. J. (1977a). Statistical constraints on synaptic plasticity. *Journal of Mathematical Biology, 69*, 385–389.

Sejnowski, T. J. (1977b). Storing covariance with nonlinearly interacting neurons. *Journal of Mathematical Biology, 4*, 303–321.

Simon, H. A. (1967). Motivational and emotional controls of cognition. *Psychological Review, 74*, 29–39.

Sloman, A. (1999). Review of "Affective computing". *AI Magazine*, 127–133.

Sloman, A. (2001). Varieties of affect and the CogAff architecture schema. In C. Johnson (Ed.), *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing* (pp. 39–48). Heslington, York, England: AISB.

Sutton, R. S., & Barto, A. G. (1981). Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review, 88*(2), 135–170.

Taatgen, N. A. (1997). A rational analysis of alternating search and reflection strategies in problem solving. In *Proccedings of the 19th Annual Conference of the Cognitive Science Society* (pp. 727–732). Mahwah, NJ: Lawrence Erlbaum Associates.

Taatgen, N. A. (2001). A model of individual differences in learning air traffic control. In E. M. Altmann, A. Cleeremans, C. D. Schunn, & W. D. Gray (Eds.), *Proceedings of the Fourth International Conference on Cognitive Modeling* (pp. 211–216). Mahwah, NJ: Lawrence Erlbaum.

Thayer, R. E. (1978). Toward a psychological theory of multidimensional activation (arousal). *Motivation and Emotion, 2*, 1–34.

Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science, 185*, 1124–1131.

Tversky, A., & Kahneman, D. (1981). The framing of decisions and the psychology of choice. *Science, 211*, 453–458.

Wine, J. (1971). Test anxiety and direction of attention. *Psychological Bulletin, 76*, 92–104.

Wood, D., & Middleton, D. (1975). A study of assisted problem–solving. *British Journal of Psychology, 66*(2), 181–191.

Yerkes, R. M., & Dodson, J. D. (1908). The relation of strength of stimulus to rapidity of habit formation. *Journal of Comparative Neurology and Psychology, 18*, 459–482.

Zajonc, R. (1980). Feeling and thinking: Preferences need no inferences. *American Psychologist, 35*, 151–175.

APPENDIX A

Code of the Dancer Model Explained

A.1 Global Parameters Settings

Many parameters in the model are set to the default values of ACT–R. Below are the settings used in all model modifications.

```
(sgp
 :er  t       ;;; enable randomness
 :era t       ;;; enable rational analysis
 :ut nil      ;;; utility threshold
 :ol t        ;;; optimised learning
 :pl t        ;;; parameters learning
)
```

Note that utility threshold (parameter `:ut`) set to `nil` means that all production rules, no matter how small their expected gains are, will be considered by the ACT–R architecture. Other global parameters, such as the goal value (`:g`), expected gain noise s (`:egs`), activation and production strength learning d (parameters `:bll` and `:sl`) can have different setting (see Chapter 4).

A.2 Declarative Memory

The model starts with the initial declarative knowledge, but it also can add new chunks during the run. In this section all the chunk–types used in the model are explained along with the chunks the model contains initially.

Chunks in ACT–R are classified by their type. Each type defines the fixed number of slots, which are usually used to describe properties of a particular class of objects. ACT–R allows chunk–types to inherit properties (slots) from their parent chunk–types using the

:include command. The description of chunks will be provided according to their types.

A.2.1 Perception

One special chunk holds the outputs of the sensory system:

```
(sensors ISA perception)
```

The separate type *perception* of this chunk ensures that only production rules from the perceptual system will be considered by the ACT–R architecture when the sensors chunk is in focus. In the current implementation the perceptual system has only two sensors, each represented by one slot:

```
(Chunk-Type perception vision skin)
```

Slot vision holds the name of a visual chunk representing an image of the object right ahead. Slot skin holds the value of external stimulation. Both slot values are supplied to the model from the simulation based on where and how the mouse is located.

Visual properties of objects are represented by chunks of type *image*:

```
(Chunk-Type image image)
```

The only slot of the image chunk is used by other chunk–types, which inherit the image type. Images are patterns on the basis of which objects can be identified. In the model they are represented by different chunks of type image. Thus, every object in the memory can have an image chunk associated with it. Below are the image chunks used by the model:

```
(black ISA image)
(white ISA image)
(black-white ISA image)
(left-to-main  ISA image)
(right-to-main ISA image)
(w-wall-i ISA image)
(e-wall-i ISA image)
(s-wall-i ISA image)
(n-wall-i ISA image)
(sep-wall-i ISA image)
```

A.2.2 Recognition

Recognition of visual objects is performed through a dedicated *recognition buffer* chunk:

```
(buffer ISA recognition)
```

This buffer has only one slot *object* defined by its chunk–type:

```
(Chunk-Type recognition object)
```

The value of the slot can either be `nil` or a chunk in the long–term memory representing the object, that has been recognised. Recognition system is used in the model mainly to reactivate the chunk each time the object is in the focus of the visual system. The idea is that if an object is new, then a new object chunk is created. If an object is recognised based on its image, then the activation of a chunk representing the object increases (see equation (2.7)).

A.2.3 Action

Commands to the motor system for performing actions are represented by chunks of a special type *action*. For example, a command to step forward is represented by a chunk:

```
(goal ISA action forward t)
```

The separate chunk–type ensures that only production rules from the action system will be considered by ACT–R when an action chunk is in focus. The chunk–type action allocates one slot for each type of action:

```
(Chunk-Type action forward backward left right)
```

Only four types of actions are implemented in the model: step forward, step backward, turn left, and turn right.

A.2.4 Space and Time

```
(Chunk-Type (point (:include image)) x y)
```

The chunk–type *point* is used to represent a point in the simulated world. It includes the *image* inheriting an `image` slot, which can hold an image chunk. The other two slots `x` and `y` hold the coordinates of a point with respect to the top left corner of the main window in the simulation. The model can be extended to three dimensions by adding a third slot, `z`. The coordinates are used by the simulation to perform specific calculations, such as to calculate the relative direction and distance from the mouse to an object.

```
(Chunk-Type (location (:include point)) direction distance time)
```

The chunk–type *location* represents relative from the mouse point of view location of a point in space and time. The type extends the *point* chunk–type, thus the location type inherits the x and y slots relating every location in the model to a physical point in the simulation. The coordinates are used by the simulation to return the values of the `direction` and `distance` slots. These two slots hold the relative location of a point.

The model is organised in such a way that precision of the location should not necessarily be perfect. Because the information about locations is used mainly for locomotion of the mouse (steps and turns), the precision of the distance and direction is derived from the precision of the movements.

Slot `distance` holds the number of steps to a point, where a step is the smallest distance a mouse can move by (the default value is 10 pixels). So, if a point is 43 pixels ahead of the mouse, then the distance to it will be equal to exactly four steps. Distance to any point within 10 pixels away from the mouse is alway considered zero.

Slot `direction` holds the number of smallest turns the mouse should do to face a point. The smallest turn defines the maximum number of turn moves necessary to make a full 360° turn. In the model this number is set by default to four, and it makes the smallest angle a mouse can turn by equal 90°. If the location is ahead of the mouse, then the direction to it is considered zero. If the location is on the right, then the direction is set to 1, and if it is on the left, then the direction is -1. Direction 2 means that the point is located behind. Such a rough precision is enough for the purposes of this study, and even at this precision quite complex movements and trajectories of the mouse can be observed in the model (see Figure 3.4). The precision of movements can be increased, if necessary.

Slot `time` is needed to distinguish between location at the current time moment and locations in the future or past. Zero value in the time slot ($t = 0$) means that the location describes the current moment. If location describes future or past, then the value of the time slot is simply `nil`.

A.2.5 Objects

```
(Chunk-Type (object (:include location)) danger in)
```

Chunks of type *object* are used in the model as internal representations of objects in the simulated world. The chunk–type extends *location* type, so description of any object includes its image and location in space and time. Thus representations of objects in the model may

describe not only their current states, but also future or past states.

Because the location type extends points, every object has x and y coordinates. These are the coordinates of geometrical centre of the corresponding object in the simulation. The simulation uses these coordinates to return the information about the relative distance and direction to an object. One may argue that a single point cannot provide enough information to describe an object's location, especially if it is a large object such as a room. Ideally, the whole set of points an object consists of should be considered. For the purposes of the described model, however, the location of geometric centre of an object proved to be sufficient.

Slot `image` inherited from the point type holds visual characteristic of an object. Its value is the name of an image chunk representing visual properties of the object, such as an image or a colour.

Slot `danger` is included into the model to represent a possibly negative attitude towards an object, and it can be used to model the so called *somatic markers*: if in the past experience an object was identified as dangerous, then once it is in focus again it may activate some subsystems (for example, it could increase arousal on alert state).

Slot `in` is used to describe the relation of an object to other objects. Particularly, if an object is spatially located inside another object, then the value of the slot `in` of the chunk representing this object will point to the chunk representing the container object. For example, if the mouse is in the left box, then the value of `in` slot of a chunk representing the current state of the mouse will point to the white box chunk. Note that such knowledge representation scheme can be used to describe complex objects that have several smaller objects as their components: every simple object, which is a component of a more complex object, would have the same value in its `in` slot: the name of the container object's chunk. Together these chunks may represent components of a more complex object.

Below are chunks representing the simple objects of which the discrimination chamber consists:

```
(unknown  ISA object)
(w-wall   ISA object image w-wall-i)
(e-wall   ISA object image e-wall-i)
(s-wall   ISA object image s-wall-i)
(n-wall   ISA object image n-wall-i)
(sep-wall ISA object image sep-wall-i)
```

Note that the model can start without the chunks above and learn new representations by itself. However, this sometimes results in learning too many chunks for one object if the model fails to recognise it. In order to make the results of the model more predictable in many runs the model starts with the set of necessary chunks.

A.2.6 Special Classes of Objects

Specific classes of objects are represented in the model by chunks of types extending the type *object*. These types have additional slots to describe the specific properties of the subclasses of objects they represent.

Self States

The most important class of objects in the model memory is the internal representation of the mouse itself. These representations describe the current, future or past states of the mouse. The states in the model are objects described by chunks of a special type *self*:

```
(Chunk-Type (self (:include object))
            ahead left right behind feel)
```

In addition to the standard slots of type object, the self chunk–type includes information about what objects are ahead, behind, and on the left or right sides. The values of these four slots are used to form the environment object that the mouse is currently in. Slot `ahead` receives its value during the perceptual cycle. Then, if the mouse turns left or right, the production system is designed in such a way that the value of the ahead slot is passed to the corresponding direction slot (see productions `turn-left` and `turn-right`). One may see these four direction slots as an abstraction of the place–cells in the hippocampus.

The `feel` slot is designed to represent the subjective description of comfort. It determines whether the mouse feels good or bad at the current moment. There may be many factors and objectives involved in this property, but in the current model the value of the slot is determined only by the skin receptors of the mouse.

Declarative memory of the model always contains at least one chunk of the *self* type, and this is the chunk representing the current state of the mouse:

```
(self0 ISA self distance 0 direction 0 time 0
       ahead unknown left unknown right unknown behind unknown
       feel  nil)
```

One can see that the location of this chunk describes zero distance in both space and time.

Environments

The rooms or boxes the mouse can move around are represented by chunks of the *environment* type extending the *object* type:

```
(Chunk-Type (environment (:include object))
            ahead left right behind other exit)
```

Slots `ahead`, `left`, `right`, and `behind` describe the objects, such as walls, surrounding the mouse. The collection of the surrounding objects comprise the current environment the mouse is in. A chunk of this type can be created once the mouse has seen all the objects around, that is when values of the slots `ahead`, `left`, `right`, and `behind` of the chunk *self* are not set to `unknown`. The mouse model can create a new environment chunk if the room is visited for the first time. The model has the ability to recognise the environment if it has previously visited the room.

Slot `other` in the environment chunk–type points to a chunk representing the complement of the environment. In general, if A is the current environment, and U is the universe, then the complement of A is $\bar{A} = U \backslash A$. Each environment chunk in the model has a chunk denoting its complement. The complement is necessary to describe the goal to escape the current box. In this case the goal is a state chunk with the complement chunk as the value of the `in` slot. For example, the chunk `not-main-box` is the complement of the `main-box` (in our case it is just a union of left and right boxes). So, an intention of the mouse to escape from the main box will be represented by a goal chunk with the `not-main-box` as the value of `in` slot.

Slot `exit` points to an object representing one or a collection of several exits from the environment.

In the current model there are initially three environments representing the three boxes in the simulation, and three environments representing their complements:

```
(left-box ISA environment
      image  Left-Box
      ahead  N-Wall
      left   W-Wall
      right  Sep-Wall
      behind Left-To-Main
      other  Not-Left-Box
      exit   Left-To-Main)
```

```
(right-box ISA environment
      image  Right-Box
      ahead  N-Wall
      left   Sep-Wall
      right  E-Wall
      behind Right-To-Main
      other  Not-Right-Box
      exit   Right-To-Main)
```

```
(main-box ISA environment
      image  Main-Box
      ahead  Wayout
      left   W-Wall
      right  E-Wall
      behind S-Wall
      other  Not-Main-Box
      exit   Wayout)

(not-main-box  ISA environment other Main-Box)
(not-left-box  ISA environment other Left-Box)
(not-right-box ISA environment other Right-Box)
```

Exits

Exits are objects of chunk–type *exit*:

```
(Chunk-Type (exit (:include object)) from to)
```

Slots `from` and `to` contain names of the environments that the exit connects. Note that the exit chunks describe the direction, so although two environments may be connected by only one door there may be two exit chunks referring to it depending on the direction.

There are two doors between the three boxes in the simulation, so there are four exit chunks:

```
(from-main-left ISA exit
      image Black
      from  Main-Box
      to    Left-Box)

(from-left  ISA exit
      image Left-To-Main
      from  Left-Box
      to    Main-Box)
```

```
(from-main-right ISA exit
      image White
      from  Main-Box
      to    Right-Box)

(from-right ISA exit
      image Right-To-Main
      from  Right-Box
      to    Main-Box)
```

A.2.7 Choice

An important part of the model is dedicated to choosing from two or more opportunities. A choice between alternatives is represented in the model by a chunk of type *choice*:

```
(Chunk-Type (choice (:include image)) first second chosen)
```

One can see that the choice type inherits slot `image` from image type. Thus, a choice, as a single object, can have a visual representation. Slots `first` and `second` point to the alternative objects. Note that if a choice is made from more than two opportunities, then in this situation a choice can still be represented by a choice chunk: the choice between the first and the other objects.

Slot `chosen` holds the chosen chunk after the choice has been made. If the value is `nil`, then it indicates that the choice has not been made yet.

The model uses one chunk of type *choice* to represent the two alternative escape doors from the main box:

```
(wayout ISA    choice
        image  Black-White
        first  From-Main-Left
        second From-Main-Right)
```

The above chunk representing a union of two exits from the main box is in the value of the `exit` slot of the `main-box` environment chunk.

A.3 Procedural Memory

Production rules can be classified by the type of the goal they operate on, as well as by the rule's functionality. In this section the production rules of the model are explained and classified into small groups. ACT–R production compilation mechanism allows the model to learn new rules during the model run. An example of a rule learned by the model is presented in the end of this section.

A.3.1 Perception and Recognition

```
(p Perceive
   =goal>
     ISA         self
   =sensors>
     ISA         perception
     vision      nil
     skin        nil
==>
   !push!        =sensors
)
```

This production rule starts the perception cycle of the model. The *self* type of the goal ensures that the model enters the perception cycle only when the focus is a state chunk (i.e. of type *self*). The constraint chunk of type *perception* should have all the slot values set to `nil`.

```
(p See-Feel
   =goal>
     ISA         perception
     vision      nil
     skin        nil
   =buffer>
     ISA         recognition
==>
   !bind!  =image (look-ahead)
   !bind!  =feel  (feel-skin)
   =goal>
     vision      =image
     skin        =feel
   =buffer>
     object      nil
)
```

This rule calls two functions from the simulation, which return values for specific types of sensors. Function `look-ahead` returns the name of a chunk representing the image of an object ahead, and function `feel-skin` returns the value of external stimulation. The production also resets the recognition buffer.

```
(p New-Feeling
   =goal>
     ISA         perception
     skin        =feel
   =self>
     ISA         self
     time        0
   - feel        =feel
==>
   =self>
     feel        =feel
)
```

This rule simply transfers a new value from skin sensors into the feel slot of the chunk representing the current state. Note that any value larger than zero is considered by the model as an aversive stimulus.

```
(p Start-Recognition
   =goal>
     ISA         perception
     vision      =image
   =buffer>
     ISA         recognition
     object      nil
==>
   !push!        =buffer
)
```

After the image ahead has been acquired, this rule starts the recognition process.

```
(p Recognise-Object
   =goal>
     ISA         recognition
     object      nil
   =sensors>
     ISA         perception
     vision      =image
   =image>
     ISA         image
   =object>
     ISA         image
     image       =image
==>
   =goal>
     object      =object
   !pop!
)
```

The production attempts to retrieve an object chunk with the same visual properties as the object ahead. If the object has been recognised, the name of the object is placed into the recognition buffer, which marks the end of the recognition process.

```
(p New-Ahead
   =goal>
     ISA         perception
     vision      =image
   =buffer>
     ISA         recognition
     object      =object
   =self>
     ISA         self
     time        0
     ahead       unknown
==>
   =self>
     ahead       =object
)
```

After the object ahead has been recognised, the rule puts the chunk representing the object into the ahead slot of the current state chunk. An additional constraint is that the object ahead should be initially unknown. Different rules (see below) treat the situations when it is not the case.

```
(p New-Environment
   =goal>
     ISA         perception
     vision      =image
   =buffer>
     ISA         recognition
     object      =object
   =self>
     ISA         self
     time        0
   - in          nil
   - ahead       =object
   - ahead       unknown
==>
   =self>
     in          nil
     ahead       =object
     left        unknown
     right       unknown
     behind      unknown
)
```

This rule similarly to the New-Ahead puts the object ahead into the slot of the current state chunk, after it has been recognised. However, in this case the previous value of the ahead slot was not an unknown object and the environment was not nil. Because the object ahead is not what the dancer "expected" to see, the rule registers the change of the environment.

```
(p Perception-Completed
   =goal>
     ISA         perception
     vision      =image
     skin        =feel
   =buffer>
     ISA         recognition
     object      =object
   =self>
     ISA         self
     time        0
     ahead       =object
     feel        =feel
==>
   !pop!
)
```

This rule ends the perception cycle once the new values from the sensors have changed the corresponding slots of the current state chunk.

A.3.2 Navigation

```
(p Object-Coordinates
   =goal>
     ISA         object
     image       =image
     x           nil
     y           nil
   =image>
     ISA         image
==>
   !bind!  =x  (get-x  =image)
   !bind!  =y  (get-y  =image)
   =goal>
     x           =x
     y           =y
)
```

This rule makes a call to the simulation to return the coordinates of an object in focus, if the current values of x and y are `nil`. The simulation returns the coordinates of the geometrical centre of the object based on its unique image name. If there is no object with such a name, then the simulation returns some special coordinates (e.g. $(0,0)$) in order to avoid an infinite loop.

The two rules below return the coordinates of one of the two objects in the choice chunk. The coordinates of both objects in the choice are needed by the learning production rules when the system learns to choose based on the features of the two objects (e.g. colour or coordinates).

```
(p First-Choice-Coordinates
   =goal>
     ISA         choice
     first       =first
     second      =second
   =first>
     ISA         object
     image       =image
     x           nil
     y           nil
   =image>
     ISA         image
==>
   !bind!        =x  (get-x =image)
   !bind!        =y  (get-y =image)
   =first>
     x           =x
     y           =y
)
```

```
(p Second-Choice-Coordinates
   =goal>
     ISA         choice
     first       =first
     second      =second
   =second>
     ISA         object
     image       =image
     x           nil
     y           nil
   =image>
     ISA         image
==>
   !bind!        =x  (get-x =image)
   !bind!        =y  (get-y =image)
   =second>
     x           =x
     y           =y
)
```

```
(p Object-Location
   =goal>
     ISA         location
     x           =x
     y           =y
     distance    nil
     direction   nil
   =sensors>
     ISA         perception
   - vision      nil
==>
   !bind! =distance
          (steps-to =x =y)
   !bind! =direction
          (direction-to =x =y)
   =goal>
     distance    =distance
     direction   =direction
)
```

This rule makes a call to the functions `steps-to` and `direction-to` the point in the simulation. These functions return values to the `distance` and `direction` slots of the relative location chunk in focus.

A.3.3 Action

The rules below operate on goals of chunk–type *action* and execute the corresponding functions in the simulation. When action functions are executed, the state of the model is changed: the new coordinates of the current state chunk (`self0`) are returned, and the values of the sensors are reset to `nil` as a side–effect of the simulation functions. The latter forces the model to enter the perceptual cycle after an action has been carried out.

```
(p Step-Forward
   =goal>
     ISA         action
     forward     t
==>
   !eval!  (step-forward)
   !pop!
)
```

```
(p Step-Backward
   =goal>
     ISA         action
     backward    t
==>
   !eval!  (step-backward)
   !pop!
)
```

Left turn and right turn productions also pass the values of the `ahead`, `left`, `right`, and `behind` slots of the current state chunk.

```
(p Turn-Left
   =goal>
     ISA         action
     left        t
   =self>
     ISA         self
     ahead       =ahead
     behind      =behind
     left        =left
     right       =right
     time        0
==>
   !eval!   (turn-left)
   =self>
     ahead       =left
     behind      =right
     left        =behind
     right       =ahead
   !pop!
)
```

```
(p Turn-Right
   =goal>
     ISA         action
     right       t
   =self>
     ISA         self
     ahead       =ahead
     behind      =behind
     left        =left
     right       =right
     time        0
==>
   !eval!   (turn-right)
   =self>
     ahead       =right
     behind      =left
     left        =ahead
     right       =behind
   !pop!
)
```

A.3.4 Control Actions

The production rules in this group are creating the goals for action if the distance to the new goal state of the mouse is not zero. If the distance is zero, then the goal state is achieved and no further actions are needed. The current model uses only five production rules: one for commanding a step forward action, two rules for commanding left or right turns; one rule to pop the finished goal, and one rule for recalling and setting the last unfinished goal if such exists. More rules can be used to perform a series of actions, such as several steps forward, or combinations of steps and turns.

The `Command-Left` rule below checks if the goal state is not located ahead or on the right. Then it creates the action goal with turn left. The location slots of the goal state are set to `nil`, because after completing the action relative location of the goal state changes. The navigation system will then update the values of the new location of the goal state. The rule `Command-Right` is similar except for the opposite direction.

```
(p Command-Left
   =goal>
     ISA         self
     direction   =direction
   - direction   0
   - direction   1
   - time        0
==>
   =goal>
     distance    nil
     direction   nil
   =turn-left>
     ISA         action
     left        t
   !push!        =turn-left
)
```

```
(p Command-Right
   =goal>
     ISA         self
     direction   =direction
   - direction   0
   - direction  -1
   - time        0
==>
   =goal>
     distance    nil
     direction   nil
   =turn-right>
     ISA         action
     right       t
   !push!        =turn-right
)
```

```
(p Command-Forward
   =goal>
     ISA          self
     x            =x
     y            =y
   - distance     0
     direction    0
   - time         0
   =self>
     ISA          self
     x            =x0
     y            =y0
     time         0
   !eval! (no-obstacle =x0 =y0 =x =y)
==>
   =goal>
     distance     nil
     direction    nil
   =step-forward>
     ISA          action
     forward      t
   !push!         =step-forward
)
```

If the goal state is directly in front (`direction 0`), the distance to it is not zero, and there is no obstacle between the two points, then create a goal to step forward.

```
(p Action-Completed
   =goal>
     ISA          self
     distance     0
     direction    0
   - time         0
     feel         nil
   =sensors>
     ISA          perception
   - vision       nil
==>
   =goal>
     x            nil
     y            nil
   !pop!
)
```

This rule pops the goal state off the stack only if the direction and distance to the goal state is zero. The coordinates of the goal state are set to `nil` in order to keep the declarative memory of ACT-R cleaner (chunks with similar values of their slots simply merge together).

```
(p Action-Continue
   =goal>
     ISA         self
     time        0
   =lastgoal>
     ISA         self
   - distance    nil
   - direction   nil
   - distance    0
   - direction   0
   - time        0
   =sensors>
     ISA         perception
   - vision      nil
==>
   =lastgoal>
     distance    nil
     direction   nil
   !push!        =lastgoal
)
```

This rule can retrieve an unfinished goal state (those with non–zero distance and direction). If such a goal state exists in the memory, then reset its location and push it on the stack. This rule can be used for activation–based retrievals of unfinished goals.

A.3.5 Environment Exploration

```
(p Explore-Left
   =goal>
     ISA         self
     direction   0
     time        0
   - ahead       unknown
     left        unknown
   =sensors>
     ISA         perception
   - vision      nil
==>
   !bind!  =x  (get-self :left-x)
   !bind!  =y  (get-self :left-y)
   =turn>
     ISA         self
     x           =x
     y           =y
   !push!        =turn
)
```

If the object on the left is unknown, then set a new goal state located at the own left side (function `get-self` returns the coordinates). The navigation rules will return zero distance to this goal state, but the direction equal to -1. The following goal will be to turn left, thus facing the object. Chunk `sensors` is checked so that the rule only could fire after the perceptual cycle has completed. The two productions below are similar except that the unknown object is on the right or behind.

```
(p Explore-Right
   =goal>
     ISA         self
     direction   0
     time        0
   - ahead       unknown
     right       unknown
   =sensors>
     ISA         perception
   - vision      nil
==>
   !bind!  =x    (get-self :right-x)
   !bind!  =y    (get-self :right-y)
   =turn>
     ISA         self
     x           =x
     y           =y
   !push!        =turn
)
```

```
(p Explore-Behind
   =goal>
     ISA         self
     direction   0
     time        0
   - ahead       unknown
     behind      unknown
   =sensors>
     ISA         perception
   - vision      nil
==>
   !bind!  =x    (get-self :back-x)
   !bind!  =y    (get-self :back-y)
   =turn>
     ISA         self
     x           =x
     y           =y
   !push!        =turn
)
```

Once all the objects around have been seen the model may attempt to recognise the current environment (or create a new environment chunk). The first rule below checks if all the objects around correspond to the objects of some known environment. If this is the case, then put the environment chunk into the in slot of the current state. The recognition requires all the objects around to be in the same slots as in the environment chunk. If the dancer is facing a wrong direction and the environment cannot be recognised, the second rule makes the dancer turn around.

```
(p Recognise-Environment
   =goal>
     ISA         self
     in          nil
   - ahead       unknown
   - left        unknown
   - right       unknown
   - behind      unknown
     ahead       =ahead
     left        =left
     right       =right
     behind      =behind
     time        0
   =known-env>
     ISA         environment
     ahead       =ahead
     left        =left
     right       =right
     behind      =behind
     other       =other
   =other>
     ISA         environment
==>
   =goal>
     in          =known-env
)
```

```
(p Find-North-Left
   =goal>
     ISA         self
     in          nil
   - ahead       unknown
   - behind      unknown
   - left        unknown
   - right       unknown
     time        0
   !eval! (not (look-north-true))
==>
   !bind!  =x  (get-self :left-x)
   !bind!  =y  (get-self :left-y)
   =turn>
     ISA         self
     x           =x
     y           =y
   !push!        =turn
)
```

A.3.6 Motivations and Appraisal

Behaviour of the dancer is derived from its motivations. It is assumed that there is an ideal state the dancer is always trying to achieve. In the current model there are two objectives that define the top motivations (see Section 3.2).

```
(p Escape
   =goal>
     ISA         self
     time        0
     in          Main-Box
==>
   =escape>
     ISA         self
     in          Not-Main-Box
     feel        0
   !push!        =escape
)
```

This rule implements the first objective of the task — to escape the `Main-Box`. Note that `Not-Main-Box` is simply the complement of the main box, and it is represented in the model by a separate chunk.

```
(p Escape-Danger
   =goal>
     ISA         self
     time        0
   - in          nil
   - feel        0
==>
   =escape>
     ISA         self
     feel        0
   !push!        =escape
)
```

This rule implements the second objective of the task, and it represents an important motivation of the dancer behaviour: avoid any aversive stimulation. If the value of stimulation in the current state is not zero (slot `feel`), then create a new goal with the desired value (`feel 0`). The rule requires the dancer to be aware of the environment it is in.

The two rules below implement appraisal of achieving the first objective (to escape the `main-box`). The first rule detects the success if the dancer is no longer in the `Main-Box` and the level of stimulation is zero. On the contrary, if the dancer has escaped the `Main-Box`, but the level of stimulation is not zero, then the second production detects a failure. Note that changes in probabilities on successes and failures lead to entropy changes. The function *change-noise* here can adjust the expected gain noise according to equation (5.3).

```
(p Escape-Success
   =goal>
     ISA         self
   - time        0
     in          Not-Main-Box
     feel        0
   =self>
     ISA         self
     time        0
   - in          Main-Box
     feel        0
   =sensors>
     ISA         perception
   - vision      nil
   - skin        nil
==>
   =goal>
     x           nil
     y           nil
     direction   nil
     distance    nil
     in          nil
     feel        nil
   !pop!
   !eval!  (change-noise)
)
```

```
(p Escape-Failure
   =goal>
     ISA         self
   - time        0
   - in          Not-Main-Box
     feel        0
   =self>
     ISA         self
     time        0
   - in          Main-Box
   - feel        0
   =sensors>
     ISA         perception
   - vision      nil
   - skin        nil
==>
   =goal>
     x           nil
     y           nil
     direction   nil
     distance    nil
     in          nil
     feel        nil
   !pop!
   !eval!  (change-noise)
)
```

The two rules below implement the appraisal of achieving the second objective after the focus has returned to the *escape–danger* goal. The first rule signals the success if the level of stimulation is zero. Otherwise the second rule below registers the failure.

```
(p Escape-Danger-Success
   =goal>
     ISA         self
   - time        0
   - in          nil
   - in          Not-Main-Box
     feel        0
   =self>
     ISA         self
     time        0
     feel        0
   =sensors>
     ISA         perception
   - vision      nil
   - skin        nil
==>
   =goal>
     x           nil
     y           nil
     direction   nil
     distance    nil
     in          nil
     feel        nil
   !pop!
)
```

```
(p Escape-Danger-Failure
   =goal>
     ISA         self
   - time        0
   - in          nil
   - in          Not-Main-Box
     feel        0
   =self>
     ISA         self
     time        0
   - feel        0
   =sensors>
     ISA         perception
   - vision      nil
   - skin        nil
==>
   =goal>
     x           nil
     y           nil
     direction   nil
     distance    nil
     in          nil
     feel        nil
   !pop!
)
```

A.3.7 Escape Strategies

When the model has a goal to escape the aversive signal (the goal created by the `escape-danger` production), then it can use two strategies to achieve it. These strategies are selected by the following production rules:

```
(p Find-Better-Point
   =goal>
     ISA          self
     x            nil
     y            nil
   - time         0
     in           nil
     feel         0
   =self>
     ISA          self
     time         0
     x            =x
     y            =y
     in           =current
   =sensors>
     ISA          perception
   - vision       nil
   - skin         nil
==>
   !bind! =rx (random-xy =x =y :x)
   !bind! =ry (random-xy =x =y :y)
   =goal>
     x            =rx
     y            =ry
     in           =current
   =change-point>
     ISA          self
     x            =rx
     y            =ry
   !push!         =change-point
)
```

This rule sets a new goal with location in a random point within the current environment. The goal then causes the dancer to move into this new location. In the current model this rule always leads to a failure, because the stimulation is present in any point of the box. However, it is not known to the dancer, and because the cost of the rule is relatively small it can get selected several times before the probability has been learned. If the level of stimulation is not high enough (G is low in equation (2.1)), then the probability has to decrease significantly in order to allow the other strategy to win.

```
(p Find-Better-Environment
   =goal>
     ISA         self
   - time        0
     in          nil
     feel        0
   =self>
     ISA         self
     time        0
     in          =current
   =current>
     ISA         environment
     other       =other
==>
   =goal>
     in          =other
   =change-env>
     ISA         self
     in          =other
     feel        nil
   !push!        =change-env
)
```

This rule implements the second strategy — to escape from the current box. Although the cost of this strategy is higher than the cost of the previous rule, the model learns that it leads to a success. When the level of stimulation is high (G is high in equation (2.1)), then high probability becomes more important in the conflict resolution than the cost, and the model quickly switches to using this strategy.

A.3.8 Escaping a Box

If the goal is to escape the current box, then first the model focuses on the chunk denoting the exit from the current environment (see `Focus-On-Exit` below). The exit, however, can be a choice of two doors. The way the model makes the choice will be described later. When one particular door (chunk of type *exit*) is in the focus, then the `Go-To-Exit` rule sets a new goal to be located at this exit. The simulation is designed in such a way that the coordinates of the exit lead to the point just outside the current environment. Thus, when the dancer moves to this point it leaves the current box automatically.

```
(p Focus-On-Exit
   =goal>
     ISA         self
     in          =other
   - time        0
     feel        0
   =self>
     ISA         self
     time        0
   - in          =other
     in          =current
   =current>
     ISA         environment
     exit        =exit
==>
   !push!        =exit
)
```

```
(p Go-To-Exit
   =goal>
     ISA         exit
     x           =x
     y           =y
     direction   =dir
     distance    =dis
     from        =current
     image       =image
   =image>
     ISA         image
   =self>
     ISA         self
     time        0
     in          =current
==>
   =goal>
     direction   nil
     distance    nil
   =gotoexit>
     ISA         self
     x           =x
     y           =y
     direction   =dir
     distance    =dis
   !focus-on!    =gotoexit
)
```

A.3.9 Choosing

The exit from the main box is a choice of two doors. Choosing and learning to choose the right exit is important for the success in the task. The model starts with two simple rules for choosing. These rules have equal properties and equal chances to be chosen in the conflict resolution. During the first 20 training tests the mouse is allowed to escape the main box through any door, so the model learns similar production parameters for both rules.

```
(p Choose1st
   =goal>
     ISA         choice
     first       =first
     second      =second
     chosen      nil
   =first>
     ISA         object
   - x           nil
   - y           nil
   =second>
     ISA         object
   - x           nil
   - y           nil
==>
   =goal>
     chosen      =first
  !push!         =first
)
```

```
(p Choose2nd
   =goal>
     ISA         choice
     first       =first
     second      =second
     chosen      nil
   =first>
     ISA         object
   - x           nil
   - y           nil
   =second>
     ISA         object
   - x           nil
   - y           nil
==>
   =goal>
     chosen      =second
   !push!        =second
)
```

After the choice has been made, the dancer focuses on the selected door and then proceeds for the exit. When it reaches the location of the exit the focus of the model returns to the choice chunk, and the model can assess whether the choice made led to a success or a failure by checking the new value of the `feel` slot. The first rule below pops the choice chunk with a success if condition `feel 0` is fulfilled. Otherwise the second rule below puts the goal into the corresponding slot of a chunk of the special type *dependency*, which will be used to learn new production rules.

```
(p Choice-Success
   =goal>
     ISA         choice
     chosen      =chosen
   =self>
     ISA         self
     time        0
     feel        0
   =sensors>
     ISA         perception
   - vision      nil
   - skin        nil
==>
   =goal>
     chosen      nil
   !pop!
)
```

```
(p Choice-Failure
   =goal>
     ISA         choice
     chosen      =chosen
   =self>
     ISA         self
     time        0
   - feel        0
   =sensors>
     ISA         perception
   - vision      nil
   - skin        nil
==>
   =learn>
     ISA         dependency
     goal        =goal
   !focus-on!    =learn
)
```

A.3.10 Learning New Rules

The two production rules `choose1st` and `choose2nd` make the choice blindly without paying any attention to the features of both objects. If the positions of the white and black doors are completely random, then there is no way the dancer can figure out which door to choose. Indeed, the probabilities learned for both rules by ACT–R reflect the distribution of the successful opportunity in the simulation. If the distribution is casual, then the probabilities will be equal on average. In order to learn to choose the correct door the dancer has to create new production rules with more complex conditions paying attention to the features of the objects. The two properties to pay attention to are the location of each door (different x coordinates), and the colour feature of each door. New rules in ACT–R are created using the *production compilation* mechanism that uses a chunk of a special type *dependency*.

```
(p Learn-CX-from-Failure
   =goal>
     ISA          dependency
     goal         =choice
   =choice>
     ISA          choice
     image        =image
     first        =first
     second       =second
     chosen       =chosen
   =first>
     ISA          exit
     image        =C1
     x            =X1
     y            =Y1
     from         =from
     to           =to1
   =C1>
     ISA          image
   =second>
     ISA          exit
     image        =C2
     x            =X2
     y            =Y2
     from         =from
     to           =to2
   =C2>
     ISA          image
==>
   !bind! =other (if (eq =chosen =first) =second =first)
   !bind! =cs    (list =first =second)
   !bind! =dcs   (list =Y1 =to1 =Y2 =to2 =from)
   =choice>
     chosen       nil
   =modified>
     ISA          choice
     image        =image
     first        =first
     second       =second
     chosen       =other
                                    =goal>
                                      goal         =choice
                                      modified     =modified
                                      stack        =other
                                      constraints  =cs
                                      dont-cares   =dcs
                                    !pop!
                                    !delete!       =modified
                                    !delete!       =goal
                                )
```

The rule on the left implements two–dimensional learning, when both features (colour and position) of the doors are used. If this rule fires, it creates a new production that makes choice of an object alternative to the one chosen last, and it will pay attention to both x coordinates and colours of the two objects. Note that the colour property chunk has to be retrieved explicitly in order to be used. Thus, if the system fails to retrieve the colour chunks, then the rule will not fire and the learning will not happen. The A–model uses activations of the colour chunks to simulate the dark visual discrimination conditions.

```
(p Learn-C-from-Failure
   =goal>
     ISA          dependency
     goal         =choice
   =choice>
     ISA          choice
     image        =image
     first        =first
     second       =second
     chosen       =chosen
   =first>
     ISA          exit
     image        =C1
     x            =X1
     y            =Y1
     from         =from
     to           =to1
   =C1>
     ISA          image
   =second>
     ISA          exit
     image        =C2
     x            =X2
     y            =Y2
     from         =from
     to           =to2
   =C2>
     ISA          image
==>
   !bind! =other (if (eq =chosen =first) =second =first)
   !bind! =cs    (list =first =second)
   !bind! =dcs   (list =X1 =Y1 =to1 =X2 =Y2 =to2 =from)
   =choice>
     chosen     nil
   =modified>
     ISA          choice
     image        =image
     first        =first
     second       =second
     chosen       =other
                                    =goal>
                                      goal         =choice
                                      modified     =modified
                                      stack        =other
                                      constraints  =cs
                                      dont-cares   =dcs
                                    !pop!
                                    !delete!       =modified
                                    !delete!       =goal
)
```

This rule is similar to the one before with one exception: the new rule compiled after the dependency goal is popped will not pay attention to x coordinates of the two doors (both `=X1` and `=X2` values are in the `dont-cares` slot of the dependency chunk). Only colour information will be used in the new production rules, which is the most successful strategy.

```
(p Learn-X-from-Failure
   =goal>
     ISA          dependency
     goal         =choice
   =choice>
     ISA          choice
     image        =image
     first        =first
     second       =second
     chosen       =chosen
   =first>
     ISA          exit
     image        =C1
     x            =X1
     y            =Y1
     from         =from
     to           =to1
   =second>
     ISA          exit
     image        =C2
     x            =X2
     y            =Y2
     from         =from
     to           =to2
==>
   !bind! =other (if (eq =chosen =first) =second =first)
   !bind! =cs    (list =first =second)
   !bind! =dcs   (list =Y1 =C1 =to1 =Y2 =C2 =to2 =from)
   =choice>
     chosen     nil
   =modified>
     ISA          choice
     image        =image
     first        =first
     second       =second
     chosen       =other
                                    =goal>
                                      goal         =choice
                                      modified     =modified
                                      stack        =other
                                      constraints  =cs
                                      dont-cares   =dcs
                                    !pop!
                                    !delete!       =modified
                                    !delete!       =goal
                                )
```

This rule is similar to the two rules above, but here the new rule compiled will not pay attention to the colour features of the two doors (both =C1 and =C2 values are in the dont-cares slot of the dependency chunk). Only position information will be used in the new production rules, which is not a good strategy in this task.

A.3.11 Example of a Rule Compiled by the Model

```
(p Choice8041
   =goal>
      isa CHOICE
      image =image
      first =first
      second =second
      chosen nil
   =first>
      isa EXIT
      image =image1
      direction nil
      distance nil
      time nil
      danger nil
      in nil
   =second>
      isa EXIT
      image Black
      direction nil
      distance nil
      time nil
      danger nil
      in nil
==>
   =first>
      isa EXIT
      image =image1
      direction nil
      distance nil
      time nil
      danger nil
      in nil
   =goal>
      chosen =first
   !push! =first
)
```

The rule on the left was compiled by ACT-R after the rule `Learn-C-from-Failure` had fired. The rule is very similar to the `choose1st` or `choose2nd` rules except that this rule pays attention to the visual property of the objects, and if the value is `Black`, then choose another object.

A.3.12 Final Production

```
(p Have-A-Rest
   =goal>
     ISA         self
     time        0
     x           =x
     y           =y
   - in          nil
   - in          Main-Box
     feel        0
   =sensors>
     ISA         perception
   - vision      nil
   - skin        nil
==>
   !bind! =nx (final-xy =x =y :x)
   !bind! =ny (final-xy =x =y :y)
   !Output!  ("Have a nice day!")
   !pop!
   =relax>
     ISA         self
     x           =nx
     y           =ny
   !focus-on!    =relax
)
```

When the current state indicates that both objectives are achieved, then this rule ends the model run by removing the current state goal chunk from the stack. Finally it sets the goal to locate the mouse in the middle of the current environment.

A.4 Parameters Settings

```
(spp
 (escape-failure        :failure t)
 (escape-danger-failure :failure t)
 (choice-failure        :failure t))
```

The three productions above have explicit flag `failure`. It means that when one of these production fires the goal is removed with the failure flag. Thus, it will be learned that a rule, which set the goal initially, led to the failure.

Most of the production parameters can be learned by the model. However, the model contains rules, which implement a very basic knowledge, that a mouse should have used all its life. Thus, the values production strengths and probabilities of these productions should have been learned already and must not be affected significantly by the learning mechanisms. The learning influences mainly the new rules created during the model run. The code below sets the values of parameters of production rules according to these assumptions.

First, the probabilities of productions `choose1st` and `choose2nd` are set to .5 and they change slowly. These rules represent the random choice strategy between two rules.

```
(spp
  (choose1st :eventual-successes 5000 :eventual-failures 5000)
  (choose2nd :eventual-successes 5000 :eventual-failures 5000))
```

Second, the strengths of productions are set. There are two ways to do it in ACT-R, depending on whether the strength learning mechanism is switched on or not. Note that rules here are separated into three groups: perception, action, and cognition (all other rules).

```
(if (car (sgp :sl))
    (progn
      (spp :references 10000 :creation-time -10000000)

      (let ((perception '(perceive see-feel
                          start-recognition recognise-object
                          new-ahead new-feeling new-environment
                          perception-completed object-coordinates
                          first-choice-coordinates
                          second-choice-coordinates
                          object-location recognise-environment))
```

```
          (action '(step-forward step-backward turn-left turn-right)))

        (loop for p in perception do
          (spp-fct (list p :references     10000000
                           :creation-time -10000000)))

        (loop for p in action do
          (spp-fct (list p :references     1000000
                           :creation-time -10000000)))))

  (spp :strength 10.0))
```

Finally, some special chunks used by the model should not require long retrieval times. These chunks have very high activations and do not affect the latencies. The activations are set using the code below. Again, the method depends on whether the base–level learning mechanism is turned on.

```
(if (car (sgp :bll))
    (progn
      (set-base-levels
       (self0   11000000 -10000000)
       (sensors 10000000 -10000000)
       (buffer  10000000 -10000000)
       (unknown  1000000 -10000000)
       (black    1 0.0)
       (white    1 0.0) ))
  (progn
    (set-all-base-levels 10.0)
    (set-base-levels-fct
     (list (list 'black *base-level*)
           (list 'white *base-level*)))))
```

APPENDIX B

Some Modifications to the ACT–R Architecture

The changes in the ACT–R architecture mentioned in Section 2.4 are implemented using two hook functions. These hooks are set in the header of the model:

```
(setf *cycle-hook-fn*  (add-hook *cycle-hook-fn* 'null-all-failures))
(setf *firing-hook-fn* (add-hook *firing-hook-fn* 'null-z-n-1))
```

where add-hook function can add a new function to an existing hook:

```
(defun add-hook (old-hook new-hook)
  "Adds a function to an existing hook"
  (if (null old-hook)
      (lambda (x) (funcall new-hook x))
    (lambda (x) (funcall old-hook x) (funcall new-hook x))))
```

The first hook function disables the goal–discounting mechanism:

```
(defun null-z-n-1 (&optional instantiation)
  (declare (ignore instantiation))
  (setf *z-n-1* *g*))
```

The second hook function removes the number of *failures* on every cycle:

```
(defun null-all-failures (&optional instantiation)
  (declare (ignore instantiation))
  (dolist (p *procedural-memory*)
     (setf (production-failures (cdr p)) (list 0.0))))
```

In addition, the number of *successes* is set in the model to a high value in order to minimise the possible effect of subprobability q (see Section 2.4).

```
(spp :successes 100000)
```

APPENDIX C

Implementation of the OPTIMIST Conflict Resolution for ACT–R

Below is the code of the first implementation of the OPTIMIST conflict resolution algorithm for ACT–R. It has been tested on a model with ACT–R Version 4. The algorithm works as an overlay to the ACT–R architecture, and it can co–exist with the standard mechanism.

C.1 Loading Instructions

The OPTIMIST conflict resolution overlay requires ACT–R code to be loaded because the code uses some of its functions.

```
(load "optimist-cr" :if-source-newer :compile)
```

The following command should be added into the header of the model after the `(clear-all)` command:

```
(setf *conflict-set-hook-fn*
                      (add-hook *conflict-set-hook-fn* 'optimist-cr))
(setf *firing-hook-fn* (add-hook *firing-hook-fn* 'reinforce-start))
(setf *firing-hook-fn* (add-hook *firing-hook-fn* 'new-z-n-1))
(setf *cycle-hook-fn*  (add-hook *cycle-hook-fn* 'reinforce-end))
```

One can see that there are four functions called by three hook functions on every cycle. These functions will be explained below. By setting the global variable `*optimist*` to `nil` it is possible to switch off the OPTIMIST mechanism.

C.2 Global Variables

```
(defvar *optimist* t)
```

Used as a predicate: If `t`, then use optimist conflict resolution.

```
(defvar *min-exp-gain-noise* nil)
```

If a number, then adds noise to instantiations in the conflict set. Similar to `:egs` parameter of ACT–R (`*exp-gain-noise*`).

```
(defvar *min-c-noise* nil)
```

If a number, then adds noise to instantiations calculated as a proportion of their expected costs. Typical values $0.01 \sim 0.2$. For example, if a production has expected cost $\bar{C} = 10$, then noise will be added with variance 10 multiplied by `*min-c-noise*`.

```
(defvar *minimal-cost* *default-action-time*)
```

Defines the smallest expected cost possible.

```
(defvar *reward* nil)
```

If a number, then subtract it from the efforts on successful completion of the goal.

```
(defvar *penalty* nil)
```

If a number, then add it to the efforts when the goal is completed with failure. Typical values $1 \sim 50$.

C.3 Functions

C.3.1 Service Functions

```
(defun unwrap (list)
  (and list (if (not (listp list)) list (unwrap (car list)))))
```

Used to access the values of parameters in ACT–R, which are normally sorted in lists.

```
(defun add-hook (old-hook new-hook)
  (if (null old-hook)
      (lambda (x) (funcall new-hook x))
    (lambda (x) (funcall old-hook x) (funcall new-hook x))))
```

Adds a function to an existing hook function.

C.3.2 Computations for Individual Productions

```
(defun p-all-efforts (p)
  (+ (unwrap (production-efforts p))
     (unwrap (production-eventual-efforts p))))
```

This function returns all efforts spent executing the production. Calculates t in equation (6.7).

```
(defun p-all-uses (p)
  (let ((*command-trace* nil))
    (if (numberp (unwrap (sgp-fct (list :sl))))
        (1+ (unwrap (production-references p)))
      (+ (unwrap (production-eventual-successes p))
         (unwrap (production-eventual-failures  p))))))
```

Returns the number of times the production was used. Corresponds to $i + 1$ number in equation (6.9).

```
(defun p-e-cost (p)
  (compute-costs (production-eventual-successes p) (list 0.0)
                 (list (p-all-efforts p))))
```

Returns the expected cost $\bar{C}$ of the production by equation (6.7).

```
(defun r-cost (mean)
  (let* ((c (if (> mean 0) mean *minimal-cost*))
         (c-min (* 0.0001 2 c))
         (c-max (* 0.9999 2 c)))
    (max c-min (min (random c-max) c-max))))
```

Returns random cost with a given expected value. Corresponds to $\xi(\bar{C})$ in equation (6.9).

```
(defun p-r-e-cost (p)
  (let* ((c (p-e-cost p))
         (i (p-all-uses p))
         (r (/ (+ (* (1- i) c) (r-cost c)) i)))
    (and (numberp *min-exp-gain-noise*)
         (incf r (noise *min-exp-gain-noise*)))
    (and (numberp *min-c-noise*)
         (incf r (noise (* c *min-c-noise*))))
    r))
```

Returns the random estimated cost $\tilde{C}$ of the production calculated by equation (6.9). In addition, if `*min-exp-gain-noise*` or `*min-c-noise*` are numbers, then adds the corresponding noise to the production.

C.3.3 Main functions

```
(defun optimist-cr (&optional conflict-set)
  (and *optimist* conflict-set
       (let* ((is conflict-set)
              (ps (mapcar #'(lambda (x)
                              (instantiation-production x)) is))
              (ecs (mapcar #'p-r-e-cost ps)))
         (loop for i  in is
               and ec in ecs do
               (setf (instantiation-gain i) (- *g* ec)))
         (setf is (sort is #'(lambda (x y)
                               (> (instantiation-gain x)
                                  (instantiation-gain y)))))
         is)))
```

This is the main OPTIMIST conflict resolution function. The argument is the standard conflict set. The function sorts the set according to random estimated costs of the productions and returns the new conflict set.

```
(defun new-z-n-1 (&optional instantiation)
  (and *optimist* instantiation
       (let ((p (instantiation-production instantiation)))
         (setf *z-n-1* (p-e-cost p)))))
```

This function implements the goal–discounting mechanism according to the estimated cost of the production that is about to fire. This function, however, is not necessary and is added for analysis purposes.

```
(defun reinforce-start (i)
  (and *optimist*
       (numberp *reward*)
       (and i (production-success (instantiation-production i)))
       (decf *time* *reward*))
  (and *optimist*
       (numberp *penalty*)
       (and i (production-failure (instantiation-production i)))
       (incf *time* *penalty*)))
```

If the goal is popped with success or failure flag, then this function adds a reward or penalty respectively to the standard ACT–R time before the parameters learning occurs (learning the new cost of production that set the goal).

```
(defun reinforce-end (i)
  "Removes reward-penalty from *time* after learning"
  (and *optimist*
       (numberp *reward*)
       (and i (production-success (instantiation-production i)))
       (incf *time* *reward*))
  (and *optimist*
       (numberp *penalty*)
       (and i (production-failure (instantiation-production i)))
       (decf *time* *penalty*)))
```

This function removes the extra effort (reward or penalty) after the parameters learning has occurred. Thus, the time in ACT–R remains unchanged.

APPENDIX D

Posterior Poisson Distribution

If λ is not known, then in order to estimate its value after observing n events we need to know the posterior probability distribution $P(\lambda \mid n)$, which can be obtained from the joint and prior probabilities of λ and n using Bayes' formula:

$$P(\lambda \mid n) = \frac{P(n,\lambda)}{P(n)} = \frac{P(n \mid \lambda)P(\lambda)}{P(n)} \ .$$

Here $P(n,\lambda)$ is the joint probability of n and λ, $P(n \mid \lambda)$ is the likelihood probability of n for given λ, and $P(\lambda)$, $P(n)$ are the prior probabilities of λ and n.

In our case the likelihood probability $P(n \mid \lambda)$ is the probability of observing n events for a given mean count rate λ and parameter t, and it is given by the Poisson distribution:

$$P(n \mid \lambda) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \ .$$

The prior distribution $P(\lambda)$ is not known, but we do know that the rate must be positive: $\lambda \in [0,\infty]$. Let us assume the following hypothesis

$$P_\varepsilon(\lambda) = ce^{-\varepsilon\lambda} \ ,$$

where c is a constant and ε is small and positive. This hypothetical distribution has the following properties: when $\varepsilon \to 0$, the probability of a particular λ becomes zero, which makes all the values of λ on $[0,\infty]$ equally probable. On the other hand, if $\varepsilon > 0$, then it means that very high rates λ are less likely than smaller rates. One can also notice that $c = \varepsilon$ due to the normalisation:

$$\int_0^\infty P_\varepsilon(\lambda)\mathrm{d}\lambda = \int_0^\infty ce^{-\varepsilon\lambda}\mathrm{d}\lambda = \frac{c}{\varepsilon} = 1 \ .$$

So, we can write

$$P_\varepsilon(\lambda) = \varepsilon e^{-\varepsilon\lambda} .$$

Using this hypothesis we can find the joint distribution:

$$P_\varepsilon(n,\lambda) = P(n \mid \lambda)P_\varepsilon(\lambda) = \frac{(\lambda t)^n}{n!}e^{-\lambda t}\varepsilon e^{-\varepsilon\lambda} = \varepsilon\frac{(\lambda t)^n}{n!}e^{-\lambda(t+\varepsilon)} .$$

The prior distribution of n can be found by integrating the above distribution over all λ:

$$P_\varepsilon(n) = \int_0^\infty P_\varepsilon(n,\lambda)\mathrm{d}\lambda = \varepsilon\frac{t^n}{n!}\int_0^\infty \lambda^n e^{-\lambda(t+\varepsilon)}\mathrm{d}\lambda = \frac{\varepsilon}{t+\varepsilon}\left(\frac{t}{t+\varepsilon}\right)^n .$$

The posterior probability now can be written using the above joint probability $P_\varepsilon(n,\lambda)$ and prior $P_\varepsilon(n)$:

$$P_\varepsilon(\lambda \mid n) = \frac{P_\varepsilon(n,\lambda)}{P_\varepsilon(n)} = (t+\varepsilon)\frac{(\lambda(t+\varepsilon))^n}{n!}e^{-\lambda(t+\varepsilon)} .$$

Note that if we admit that all values of $\lambda \in [0,\infty]$ are equally probable ($\varepsilon = 0$), then

$$P_0(\lambda \mid n) = t\frac{(\lambda t)^n}{n!}e^{-\lambda t} = tP(n \mid \lambda) .$$