

A Methodology for Computational Cognitive Modelling

by
Terrence C. Stewart
M.Phil, B.A.Sc

A thesis submitted for examination
to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

Institute of Cognitive Science
Carleton University
Ottawa, Ontario

© Terrence C. Stewart, 2007



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-33512-3
Our file *Notre référence*
ISBN: 978-0-494-33512-3

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

Abstract

This thesis presents a novel methodology for computational modelling in cognitive science. It emphasizes identifying sets of models and ranges of parameter values which are equivalent to observed real-world cognitive behaviour. This equivalence is determined by a new statistical measure called relativized equivalence, which places an upper bound on the difference between the model and reality, taking into account the sampling confidence intervals. If the models are constrained to be process models, then this equivalence can also be evidence that the models follow the same algorithm as the real cognitive system.

This methodology also provides guidance for the overall process of modelling. When many models are found to be equivalent, the only way to distinguish among them is to include more empirical data. This usually involves new empirical results with different measures than originally considered. When few models (or none at all) are equivalent, this suggests that either the modelling constraints should be relaxed by removing measures or requiring less accuracy, or that new models should be developed.

To demonstrate the usefulness of this methodology, it is applied to a variety of cognitive modelling tasks. The RELACS model of repeated binary choice (Erev & Barron, 2005) is reinvestigated, showing that it is only a good model for situations with simple reward structures. In the original, standard approach, these situations were combined with many more complex situations, so this fine-grained conclusion could not be reached.

Furthermore, this analysis shows that an internal component of RELACS is vital to its

performance, in contrast with the original conclusion.

The methodology is also applied to a model of peer group interaction, demonstrating how new models can be created, including the use of existing cognitive architectures (ACT-R; Anderson & Lebiere, 1998) to guide and constrain. Finally, situations involving qualitative modelling (i.e., modelling without numerical comparison to real-world data) are explored.

The result is a method for analyzing computational cognitive models emphasizing explanation rather than prediction. This is meant to be usable by any modeller, without requiring extensive mathematical or computational skill. A software toolkit (CCMSuite) is developed to support this process.

Keywords: cognitive modelling, philosophy of modelling, cognitive architectures, statistical comparison, equivalence testing, modelling methodology, repeated binary choice, sociometric status, epistemic structures, ACT-R, Python ACT-R, CCMSuite

Acknowledgements

This dissertation could not have occurred without the unique collection of people comprising the Institute of Cognitive Science at Carleton University. Dr. Robert West, my supervisor, provided a deeply encouraging forum for the exploration of modelling ideas. These ideas were challenged and critiqued by Dr. Andrew Brook and Dr. Babak Esfandiari, who performed above and beyond the call of duty as members of my thesis committee. Their patience and insights brought clarity and relevance to what were often quite scattered thoughts.

I am also indebted to all of my colleagues in the Cognitive Science program, who let me use them as guinea pigs for my attempts at teaching modelling to non-modellers. Numerous discussions and late nights with Aryn Pyke, Matthew Rutledge-Taylor, Jacky Tweedie, Marcie Penner-Wilger, Robert Thomson, Andrea Gormley, Mark Tovey, Jennifer Schellinck, Sanjay Chandrasekharan, Luke Jerzykiewicz, David Leibovitz, Kathy Hoos, and many others have been a tremendous source of information, perspectives, and pleasure.

This process could not have been completed without the administrative support of Colleen Fulton, Lianne Dubreuil, and Dr. Andrew Brook (in his other vital role). Thank you for all of your effort and for solving all of the many problems that I created.

I cannot express how important Colleen McCutcheon has been to this work. Her enthusiasm and startling insights are a continual source of wonder for me in all areas of my life. Finally, I could not have arrived at this point without the love, support, and general preparation for everything this world has to offer provided to me by my parents, Lynn and Al Stewart. They instilled a joy of learning and a drive to understand the richness and complexity around us that is, I believe, at the heart of this work and of my life.

Table of Contents

1	Introduction.....	1
1.1	Problem Statement.....	2
1.2	Overview.....	4
1.2.1	State of the Art.....	5
1.2.2	Background.....	7
1.2.3	Contributions.....	8
2	Computational Cognitive Modelling.....	12
2.1	Cognitive Modelling.....	12
2.1.1	Origins: Beyond Behaviourism.....	12
2.2	Computational Modelling.....	14
2.2.1	Black-Box Models.....	14
2.2.2	Complete Specification.....	18
2.2.3	Models as Hypotheses.....	19
2.2.4	Modelling the Environment.....	22
2.3	A Computational Cognitive Model.....	27
2.3.1	Defining the Environment.....	28
2.3.2	Defining the Model.....	32
2.3.2.1	What The Model Does.....	34
2.3.2.2	Possible Variations.....	37
2.3.3	Evaluating the Model.....	39
2.4	But Is It Really Cognition?.....	42
2.5	Summary.....	43
2.5.1	State of the Art.....	43
2.5.2	Background.....	44
2.5.3	Contributions.....	44
3	Developing a Methodology: Equivalence.....	45
3.1	State of the Art: Three Types of Model/Reality Equivalence.....	46
3.1.1	Behavioural Equivalence.....	47
3.1.2	Algorithmic Equivalence.....	50
3.1.2.1	Complexity Equivalence.....	51
3.1.3	Structural Equivalence.....	53
3.1.3.1	Construction.....	55
3.1.3.2	Localization.....	57
3.2	Background: Interpreting Equivalence.....	58
3.2.1	Interpretation 1: Giere's "Principles".....	59
3.2.2	Interpretation 2: Thagard's "Analogies".....	61
3.2.3	Interpretation 3: Dennett's "Patterns".....	62
3.3	Contributions: How to Establish Equivalence.....	65
3.3.1	Time and Causality.....	66

3.3.2	Reductionism.....	69
3.3.3	What Should Be Equivalent.....	73
3.4	Summary.....	80
3.4.1	State of the Art.....	80
3.4.2	Background.....	81
3.4.3	Contributions.....	82
4	Developing a Methodology: Measurement.....	83
4.1	Background: Modelling in Physics.....	83
4.1.1	Computational Models.....	85
4.1.2	Numerical Comparison.....	89
4.1.3	Parameter Values.....	90
4.2	State of the Art: Numerical Comparison in Modelling.....	92
4.2.1	Correlation.....	94
4.2.2	Root Mean Squared Difference.....	99
4.2.3	The χ^2 Test.....	102
4.2.4	The t-Test and Analysis of Variance.....	107
4.2.5	The Equivalence Test.....	112
4.2.6	Bootstrap Confidence Intervals.....	116
4.2.7	Multiple Measures.....	122
4.2.7.1	Multiple Situations.....	123
4.2.7.2	Multiple Types of Measures	126
4.2.7.3	Statistics and Multiple Measures.....	127
4.3	Contributions: Measurement in Computational Cognitive Modelling.....	128
4.3.1	The Relativized Equivalence Measure.....	129
4.3.2	Parameters, Explanations, and Prediction.....	131
4.3.3	Parameter Fitting.....	137
4.3.4	Predictive Fitting.....	138
4.3.5	Parameter Ranges.....	141
4.3.6	Non-numerical Parameters.....	144
4.3.7	Model Comparisons.....	147
4.4	Summary.....	148
4.4.1	State of the Art.....	148
4.4.2	Background.....	149
4.4.3	Contributions.....	150
5	Developing a Methodology: Communication.....	152
5.1	Communication in Physics.....	152
5.2	Replication and Communication.....	153
5.3	SHRDLU: A Cautionary Tale.....	155
5.4	Complete Specification.....	156
5.5	Communication Infrastructure.....	159
5.5.1	Modelling Toolkits.....	160
5.5.1	Online Repositories.....	162
5.6	Summary.....	164
5.6.1	State of the Art.....	164

5.6.2	Background.....	164
5.6.3	Contributions.....	164
6	A Methodology for Computational Cognitive Modelling.....	166
6.1	Step 1: Experimental Situations and Measures.....	166
6.2	Step 2: Creating Models.....	167
6.3	Step 3: Evaluating Models.....	169
6.4	Step 4: Describing Models and Results.....	172
6.5	The Iterative Process of Modelling.....	174
6.6	Comparison to Other Methodologies.....	176
6.7	Carleton Cognitive Modelling Suite.....	180
6.7.1	CCMSuite Step 1: Situations and Measures.....	181
6.7.2	CCMSuite Step 2: Models.....	183
6.7.3	CCMSuite Step 3: Evaluation.....	185
6.7.4	CCMSuite Step 4: Communication.....	187
6.8	Applying and Validating the Methodology.....	187
6.9	Applying and Validating the CCMSuite Toolkit.....	189
6.10	Summary.....	191
6.10.1	Contributions.....	191
7	Basic Model Evaluation.....	193
7.1	Environment.....	193
7.2	Model.....	196
7.3	Evaluation.....	198
7.4	Summary.....	204
7.4.1	Methodology.....	204
7.4.2	Contributions.....	205
8	Parameter Variation.....	206
8.1	Changing One Parameter.....	207
8.2	Changing Two Parameters.....	215
8.3	Changing Three or More Parameters.....	219
8.4	Summary.....	226
8.4.1	Methodology.....	226
8.4.2	Contributions.....	226
9	Multiple Measures.....	228
9.1	Multiple Measures: Repeated Binary Choice.....	230
9.1.1	Many More Situations.....	236
9.2	Multiple Measures: Modelling Peer Groups.....	246
9.2.1	Popularity and Peer Groups.....	247
9.2.2	Real-World Data.....	249
9.2.3	Environment and Measures.....	251
9.2.4	A Simple Model.....	252
9.3	Summary.....	255
9.3.1	Methodology.....	255
9.3.2	Contributions.....	257
10	Model Variation.....	258

10.1	Model Variation: RELACS.....	258
10.2	Creating a New Model: Peer Groups.....	263
10.3	Summary.....	270
10.3.1	Methodology.....	270
10.3.2	Contributions.....	271
11	Modelling Architectures: Python ACT-R.....	272
11.1	Deconstructing ACT-R.....	274
11.2	Communications.....	275
11.2.1	Chunks.....	275
11.2.2	Buffers.....	276
11.2.3	Chunk Matching.....	277
11.3	Production System.....	279
11.3.1	Production Conflict Resolution.....	283
11.4	Chunk Storage.....	285
12	Using Python ACT-R.....	288
12.1	ACT-R and Repeated Binary Choice.....	288
12.1.1	Results.....	290
12.2	ACT-R, Sequential Dependencies, and Repeated Binary Choice.....	292
12.2.1	Results.....	295
12.3	ACT-R and Peer Groups.....	296
12.4	Summary.....	302
12.4.1	Methodology.....	302
12.4.2	Contributions.....	303
13	Qualitative Models.....	304
13.1	Model Exploration.....	306
13.1.1	Individual Variations in the Mathematical Model.....	307
13.1.1.1	Interpretation Bias.....	308
13.1.1.2	Behaviour Bias.....	309
13.1.1.3	Interaction Rate.....	310
13.1.1.4	Behavioural Consistency.....	311
13.1.1.5	Predisposition.....	311
13.1.1.6	Learning Rate.....	312
13.1.1.7	Patterns of Individual Variation.....	313
13.1.2	Individual Variations in the ACT-R Model.....	314
13.2	Existence Proofs.....	316
13.2.1	Learning to Create Epistemic Structures.....	317
13.2.1.1	Environment 1: Route Finding.....	319
13.2.1.2	The Q-Learning Model.....	323
13.2.1.3	Results.....	325
13.2.2	Learning to Create Internal Structures (Proto-Representations).....	328
13.2.2.1	Environment 2: Remembering State.....	328
13.2.2.2	The Q-Learning Model.....	332
13.2.2.3	Results.....	333
13.3	Summary.....	336

13.3.1	Methodology.....	336
13.3.2	Contributions.....	337
14	Conclusions.....	339
14.1	Modelling Methodology.....	342
14.2	Modelling Toolkit.....	343
14.3	Modelling Results.....	344
15	References.....	347
16	Appendix A: The Carleton Cognitive Modelling Suite.....	354
16.1	Installation.....	354
16.1.1	Installing Python.....	355
16.1.2	Installing CCMSuite.....	355
16.1.3	Configuring CCMSuite.....	357
16.2	Creating Models Environments.....	358
16.2.1	Example 1: A Blinking Light.....	359
16.2.2	Example 2: Responding to the Participant.....	361
16.2.3	Example 3: Making Measurements.....	364
16.3	Visualizing Model Runs.....	366
16.4	Distributed Execution.....	368
16.4.1	Setting up the Server.....	369
16.4.2	Adding a Model to the Server.....	370
16.4.3	Saving the Model on the Server.....	370
16.4.4	Running the Model Multiple Times.....	372
16.5	Statistical Analysis.....	374
16.5.1	Viewing the Raw Data.....	374
16.5.2	Installing Matplotlib.....	375
16.5.3	Data Analysis.....	375
16.6	Parameter Exploration.....	377
16.6.1	Defining Parameters.....	378
16.6.2	Defining an Exploration.....	379
16.6.3	Running an Exploration.....	380
16.6.4	Analyzing Single Parameters.....	381
16.6.5	Analyzing Multiple Parameters.....	385
17	Appendix B: The Models.....	388
17.1	Coin Flipping.....	388
17.2	Basic Repeated Binary Choice.....	388
17.3	Repeated Binary Choice with Parameters.....	390
17.4	Repeated Binary Choice with Multiple Measures.....	391
17.5	Peer Groups.....	397
17.6	Repeated Binary Choice using Cognitive Architectures.....	400

List of Figures

Figure 2.1: A black-box model of high-level decision making.....	17
Figure 2.2: Agent-Environment interaction.....	23
Figure 2.3: Agent-Environment interaction and environment specification.....	30
Figure 2.4: Source code for a computational model of Environment 1.....	31
Figure 2.5: Overview of the RELACS model of repeated binary choice.....	37
Figure 2.6: Comparison of human (left) and model (right) performance in three different conditions of a repeated binary choice task.....	40
Figure 4.1: Scatter plot for comparison of model and human data.....	96
Figure 4.2: Scatter plots for comparison of model and human data.....	98
Figure 4.3: Combining confidence intervals to determine the maximum likely difference between two sets of data.....	114
Figure 4.4: Eleven models with different values for parameter p compared to the experimental data from Table 4.9.....	134
Figure 4.5: Equivalence test measure for each of the models shown in Figure 4.4.....	135
Figure 6.1: The computational cognitive modelling methodology flowchart.....	176
Figure 6.2: Complete source code for an example environment demonstrating the basic timing and triggering capabilities of CCMSuite.....	182
Figure 6.3: Complete source code for an example environment demonstrating the measurement and logging capabilities of CCMSuite.	183
Figure 6.4: Complete source code for an example model demonstrating the basics of model creation and interaction with the environment in CCMSuite.....	184
Figure 7.1: Agent-Environment interaction and environment specification for exp. 2...	194
Figure 7.2: Overview of the RELACS model of repeated binary choice.....	197
Figure 7.3: Comparison of RELACS model to human data from Experiment 2.....	200
Figure 8.1: The effect on Pmax2 of adjusting parameter a over a wide range.....	209
Figure 8.2: The effect on Pmax2 of adjusting parameter a over a narrow range.....	210
Figure 8.3: The maximum likely difference between the RELACS model and human subjects.....	211
Figure 8.4: The effect on Pmax2 of adjusting parameter a over a very narrow range, and the corresponding equivalence data.....	214
Figure 8.5: The effects of adjusting parameters B , L , and K individually.....	215
Figure 8.6: Effects of varying both a and B	216
Figure 8.7: Contour plot of the effects of changing a and B	218
Figure 8.8: Effects of varying a , B , and L	220
Figure 8.9: The effects of varying all four parameters.....	222
Figure 8.10: The effects of varying all four parameters on the standard deviation of Pmax2.....	224
Figure 9.1: Agent-Environment interaction and environment specification for experiments 1, 2, and 3.....	231

Figure 9.2: The effects of varying all four parameters on the mean of Pmax1 and Pmax2 in environments 1, 2, and 3.....	235
Figure 9.3: Results for the Payoff Variability Effect measures, excluding measures #1, #7, #11, and #13.....	240
Figure 9.4: Results for the Underweighting Effect measures.....	243
Figure 9.5: Results for the Loss Rate Effect measures, excluding measures #28, #33, #34, and #40.....	244
Figure 9.6: Results for the all measures found in Figures 9.3, 9.4, and 9.5.....	245
Figure 9.7: Agent-Environment interaction for the peer group environment.....	252
Figure 9.8: The random model for peer interactions.....	253
Figure 9.9: Proportion of individuals in each of the five CDC categories.....	254
Figure 9.10: Stability of individuals in each of the five CDC categories.....	255
Figure 10.1: Equivalence for RELACS without the Fast Best Reply strategy.....	261
Figure 10.2: Equivalence for RELACS without the Loss Aversion strategy.....	261
Figure 10.3: Equivalence for RELACS without the Slow Best Reply strategy.....	261
Figure 10.4: Equivalence for RELACS with random selection between strategies.....	262
Figure 10.5: Two individuals interacting over 300 iterations of the model.....	267
Figure 10.6: The mathematical model for peer interactions.....	267
Figure 10.7: Individual and combined results for the model of peer groups.....	269
Figure 12.1: Overview of the procedural memory-based ACT-R models of repeated binary choice.....	289
Figure 12.2: Source code for an ACT-R model of repeated binary choice using the PG-C production system.....	289
Figure 12.3: Relativized equivalence of the procedural memory ACT-R model for the three categories of repeated binary choice measures.....	291
Figure 12.4: Overview of the sequential dependency and declarative memory ACT-R model of repeated binary choice.....	295
Figure 12.5: Relativized equivalence of the declarative memory ACT-R model for the three categories of repeated binary choice measures.....	296
Figure 12.6: The ACT-R model for peer interactions.....	300
Figure 12.7: Individual and combined results for the ACT-R model of peer groups.....	301
Figure 13.1: Effect size for mathematical model when adjusting interpretation bias.....	309
Figure 13.2: Effect size for mathematical model when adjusting behaviour bias.....	310
Figure 13.3: Effect size for mathematical model when adjusting interaction rate.....	311
Figure 13.4: Effect size for mathematical model when adjusting behavioural consistency.....	311
Figure 13.5: Effect size for mathematical model when adjusting initial predispositions.....	312
Figure 13.6: Effect size for mathematical model when adjusting learning rate.....	313
Figure 13.7: Summary of individual difference effects for the mathematical model.....	314
Figure 13.8: Summary of individual difference effects for the ACT-R model.....	316
Figure 13.9: Specification for the simulation environment for exploring epistemic structure generation.....	323
Figure 13.10: Overview of the Q-Learning model for learning to use epistemic structures.....	

for foraging.....	325
Figure 13.11: Agents foraging after 10, 100, and 300 time steps.....	325
Figure 13.12: Foraging rate for the epistemic structure model.....	326
Figure 13.13: The effect of varying the distance between target and home.....	327
Figure 13.14: Specification for the simulation environment for exploring internal structure generation.....	332
Figure 13.15: Overview of the Q-Learning model for learning to use internal structures for foraging.....	333
Figure 13.16: Foraging rate over time with and without internal structure formation....	334
Figure 13.17: Behaviour of internal structure learning model over various parameter settings.....	335

Index of Tables

Table 2.1: How the first rule's internal expectations of the reward for pushing the two buttons change over time given rewards.....	36
Table 4.1: Various difference measurements for comparing model and real data.....	101
Table 4.2: Example frequency data for different possible responses to the same experimental situation by 20 separate participants and 1000 model runs.....	104
Table 4.3: Example data from a study with 16 subjects.....	117
Table 4.4: Various statistics from the example data shown in Table 4.3.....	117
Table 4.5: Data from Table 4.3 resampled with replacement five times.....	119
Table 4.6: 95% bootstrap confidence intervals for various statistics of the data.....	120
Table 4.7: Example data from flipping a coin 30 times.....	133
Table 4.8: A very simple model of coin flipping, with a single parameter p	133
Table 4.9: Parameter values to be considered for the coin-flipping model.....	134
Table 4.10: Sample Data for Predictive Fitting.....	139
Table 7.1: Summary statistics for Experiment 2.....	195
Table 7.2: Summary statistics for Experiment 2.....	196
Table 7.3: Parameter settings of the best version of the RELACS model.....	197
Table 7.4: 95% Confidence Intervals for the RELACS model.....	199
Table 8.1: Parameter settings for the RELACS model to be investigated to indicate the global effect of parameter a	208
Table 8.2: Parameter settings for the RELACS model to be investigated to indicate the parameter values a which give results similar to humans.....	210
Table 8.3: Parameter settings for the RELACS model with two parameters being varied at the same time.....	216
Table 8.4: Parameter settings for the RELACS model with two parameters being varied at the same time.....	219
Table 8.5: The only two RELACS models known to differ from the human performance by less than its confidence interval for both the mean and standard deviation.....	225
Table 9.1: Data from experiments 1, 2, and 3.....	233
Table 9.2: Experimental conditions that exhibit the payoff variability effect.....	238
Table 9.3: Experimental conditions that show the effects of adjusting the magnitude of the reward.....	238
Table 9.4: Experimental conditions that show the underweighting of rare outcomes effect.....	238
Table 9.5: Experimental conditions that show the loss rate effect.....	239
Table 9.6: The decision rules for CDC sociometric classification.....	249
Table 9.7: The 95% confidence intervals indicating the percentage of people in each CDC category.....	250
Table 9.8: The 95% confidence intervals for the stability of each CDC category.....	250
Table 12.1: Experimental conditions that had to be removed.....	292

1 Introduction

The primary goal of this thesis is to determine how the tool of computer simulation can be best used within the domain of cognitive science. This includes both theoretical and practical concerns: how computational models ought to be used and what can be done to encourage their optimal use. To accomplish this, not only must the philosophical framework be developed and the computational tools be created, but they must then be applied to a variety of cognitive science topics, demonstrating the research progress possible with this methodology. The result is not only a clear research method and the accompanying tools, but also interesting novel results in modelling a variety of cognitive phenomena.

The methodology developed in this thesis is applicable to any scientific domain where the objects of study are complex enough that the predictions of a theory cannot be easily derived from a written description of that theory. This includes a broad cross-section of modern science, but all of the examples and explorations used herein come from the domain of cognitive science. The intent is that this methodology can be understood and applied by any cognitive researcher. This means that it is expected that most of this audience does not have an extensive background in computational theory or computer programming. Indeed, one of the major drives in this thesis is to enable researchers without such a background to make use of computational modelling. However, it is also expected that the audience for this thesis is aware of and appreciates the importance of a rigorous methodology for the advancement of scientific knowledge.

1.1 Problem Statement

Computational models appear throughout cognitive science. These models are a vital part of ongoing research, and are an important tool for research into any complex cognitive phenomenon. This has led to a wide variety of different approaches to using and analyzing the results of such modelling work. How models are compared to real cognitive systems, how they are validated, and how the results inform overall research programs vary from researcher to researcher.

In the most common cases, models are created that have unspecified parameters. When applying a model to a particular situation, the parameters are *fit* to that situation by adjusting them until the model's outputs are as close as possible to the empirical observations. The closeness of this fit (usually measured in terms of correlation or mean squared error) is an indication of the quality of the model. In more complex modelling approaches (e.g. Forster, 2000; Myung & Pitt, 2002), this process is enhanced by also attempting to determine how well the model generalizes. This is meant to avoid the common problem of over-fitting to the particularities of the sample data, rather than matching the model to the overall behaviour of the whole population of systems being modelled.

Other researchers have adjusted this overall process in a variety of different ways. In some cases (e.g. Anderson et al., 2004) the output of the model is used to determine whether the observed behaviour is *unlikely*, given that model. In other cases (e.g. Roberts & Pashler, 2000), the model is analyzed by examining *all possible* outputs given any parameter value. The importance of components of a model and the particular parameter

values are often investigated by removing and adjusting them to observe their impact on the overall fit (e.g. Erev & Barron, 2005), as a form of sensitivity analysis.

A common theme to all of these approaches is that they are meant to identify the *best* model out of the ones under consideration. However, this means that one model will be identified as the best even if there is insufficient evidence to be *confident* in this conclusion. Furthermore, the more complex methodologies require an array of assumptions about the underlying data distributions and various unfamiliar mathematical techniques. Finally, these methods do not provide explicit guidance as to how various modelling results can inform the researcher about the overall modelling process, indicating when more empirical evidence is needed, or when new models should be developed.

This problem addressed in this thesis is the creation of a complete methodology for cognitive modelling. This includes not only techniques for measuring how well a model matches reality, but also what features a model should have, how to adjust the set of measures used for comparison, when to develop more stringent tests, and in what situations a model should be abandoned in favour of a newly created one. In order to do this, the standard idea of identifying one best fitting parameter value for a model is abandoned, in favour of tracking large sets of *equivalent* models and parameter settings. Furthermore, the resulting methodology should be as widely accessible as possible, and make minimal assumptions. The ultimate goal is that computational modelling be an accessible tool for all cognitive scientists, regardless of their mathematical or computational background.

1.2 Overview

To begin the process of developing an overall methodology for modelling complex cognitive systems and evaluating those models, chapter 2 presents a definition of modelling in general and computational cognitive modelling in specific. An example of a current model of repeated binary choice is introduced, and will be repeatedly returned to throughout this dissertation. The next three chapters break the problem of modelling down into three very different components: Equivalence, Measurement, and Communication. Equivalence (chapter 3) deals with the philosophical problem of what it means to have a model, and in what ways it can be said to be equivalent to the system being modelled. Measurement (chapter 4) addresses the more technical problem of how to numerically compare a model to the real system. Communication (chapter 5) discusses the more pragmatic problem of how something as complex as a computational model can be effectively communicated and used within the scientific community.

In chapter 6, the results of the previous chapters are combined into the core contribution of this thesis: a clear methodology for performing computational cognitive modelling research. Since this methodology makes use of a number of non-standard processes, it is accompanied by a collection of software tools (the Carleton Cognitive Modelling Suite) which has been specifically built to support this methodology.

The remainder of the thesis (chapters 7 to 13) validates this methodology by applying it to a variety of modelling situations. These include making repeated binary choices, interacting with peer groups, and foraging for food using pheromones. Different aspects that can arise during modelling are examined in the different chapters. Chapter 8

examines the effects of parameter adjustment. Chapter 9 deals with having multiple measures. Chapter 10 makes architectural changes to the models. Chapters 11 and 12 apply this to a the cognitive modelling architecture known as ACT-R. Chapter 13 then shows how these techniques can be applied even when there is no particular real system being modelled.

At the end of each chapter, a summary is provided. These not only pick out the key aspects of the chapters, but also organize them in terms of the main thesis question. This helps organize what parts can be thought of as part of the state of the art for modelling methodology, what parts are necessary background, and what parts are the novel contributions of this thesis.

1.2.1 State of the Art

Throughout chapters 2 through 5, a variety of other researchers' contributions to the discussion of how cognitive modelling should be done are discussed. This starts with the broad recognition that computational models are required to produce quantitative predictions from complicated theories, and part of this process involves making explicit all of the assumptions underlying a theory. Specification of a model also involves making clear how the inputs and outputs to the model map onto inputs and outputs of the real system. This indicates what aspects of the cognitive process are or are not important for the model.

There is also a general recognition that useful scientific models need to be applicable in a wide set of domains. This includes both multiple experimental situations and multiple

empirical measures within each situation. These different empirical measures should include things such as error rates, reaction times, and the effects of learning, as appropriate.

The reason that reaction time measures in particular are so important is that these are the main measures capable of indicating whether the *algorithm* used by the model is the same as that used by the real system. Since there is no direct way of identifying the algorithm, its effects must be determined by examining how the reaction time changes in different experimental situations. This is the core of Pylyshyn's idea of strong equivalence (Pylyshyn, 1984).

Measuring how similar the model and the system being modelled are is a complicated task. In physics, this measure tends to deal with finding an upper bound on the difference between the predictions of the model and the experimental observations. Since the behaviours examined in cognitive modelling tend to be complex (and thus have complex probability distributions), it is desirable to attain an upper bound not just on how different the mean performance is, but also other aspects of the distribution (median, standard deviation, etc.). However, none of the standard measures used in cognitive computational modelling currently give this sort of information.

Finally, there is a wide-spread recognition among the cognitive modelling community that there is a problem of replication and communication of models. Most models are only used by the original developer of the model, and there is very little comparison between models. This may be due to the difficulty in sharing implementations, or in

communicating the actual full details of the algorithm.

1.2.2 Background

To progress from the current state of the art to the creation of a novel methodology for computational cognitive modelling, a variety of background topics are raised. The most straightforward of these background issues is the presentation of the RELACS model of repeated binary choice. This is used in chapters 2 to 5 as a concrete example for framing the modelling discussion. This raises various issues involved in modelling, such as the importance of varying aspects of a model (i.e. numerical parameters), and determining what other models are and are not equivalent to the real-world system to a given degree.

There is also a discussion of the epistemic status of models that will be familiar to many with a background in philosophy of science. A successful model can be seen as a claim that the components within the model reflect, in some meaningful way, the components of the real-world system being modelled. For some (e.g., Giere, 1999), this is not an important claim; the usefulness of a model is entirely in its predictive power, and whether the components within it can somehow be meaningfully said to actually exist within the real system is irrelevant. For others (e.g., Thagard, 1995), the fact that a model works across a broad range of situations establishes it as a component of an overall scientific analogy, and is thus as real as any other scientific analogy. Still others (e.g., Ross, 2000) claim that it is exactly those things that can be successfully modelled in this way (i.e. can be described succinctly and accurately by models) that can be thought of as being real entities. No matter which of these positions is adopted, the same methodological

requirement arises: models should be measured in terms of their minimum degree of accuracy (maximum likely error), and should be applicable to a broad range of situations.

Turning to numerical measurement, chapter 4 provides a detailed comparison of various statistical tools used to compare models and empirical observations. Of particular importance is the identification of methods which take statistical sampling into account. These do not rely on comparisons of average data values, and instead take into account confidence intervals. This is especially important in cognitive modelling, since most of the time the empirical results come from relatively few participants. By comparing to the population distribution rather than the sample distribution, problems of over-fitting are avoided, and the models' performance is characterized in a more general way. In particular, a test known as the *equivalence test*, which is generally used in comparisons in medical studies, is identified as particularly appropriate to cognitive modelling.

1.2.3 Contributions

There are two sorts of original contributions in this thesis. The first contributions come from the first half of the thesis (chapters 2-6), where the methodology is developed.

These involve theoretical considerations of how modelling should be approached, what sorts of models are needed, and how they should be evaluated and communicated. The second contributions are given in the second half of the thesis (chapters 7-13). Here, the methodology is applied to a variety of cognitive modelling situations, resulting in novel findings about the capabilities of these models.

In chapter 2, it is argued that an oft-neglected aspect of modelling is to precisely define

the environmental situation the model will be placed in. This is, in effect, a model of the experimental environment. Precisely defining the match between the environment model and the real environment is necessary for identifying exactly what aspects of the situation are being modelled. Adapting a set of suggestions for comparative studies in cognitive psychology, it is seen that the necessary details involve indicating what stimuli are presented to the model, how the model is configured to perform the particular task, and how the outputs from the model are interpreted as actions.

In chapter 3, it is made clear that models should be *process models*. That is, if a model is meant to explain how a cognitive process occurs, it should conform to the cognitive steps used by the real system. The main restriction this creates is that information within a model must not be transferred backwards in time. Doing so makes models impossible to be physically implemented, and thus not a plausible explanation of cognitive performance. Furthermore, timing information gathered from such a model can be used to establish that the model's algorithm corresponds to the real system.

As a separate contribution in chapter 3, it is noted that models make an implicit claim about the aspects of the cognitive system that they do not model. A model which is successful (to a certain degree) without involving details of some component of cognition implies that those aspects which are ignored are not important for this aspect of cognition. The accuracy of the model thus forms an upper bound on how much impact the details of ignored components may have on the overall performance.

In chapter 4, the main contribution is the development of the relativized bootstrap

equivalence measure (E_r). This is a simple, non-parametric measure that is applicable to any measure and any statistic of interest. This provides a 95% confident upper bound on the difference between the model and the system being modelled in terms of the particular measure. These equivalence measures can be combined (via scaling) into a single overall number which indicates whether the model is equivalent to the real data on *all* measures. All models which satisfy this requirement can then be classified as “good” models. Importantly, there is no sense in which one model is “more equivalent” than another: all models with $E_r < 1$ must be treated as equally plausible explanations of the cognitive phenomenon. This is an original measure, and no other measures used in cognitive modelling have these characteristics.

In chapter 5, it is noted that all of the communication and replication problems in cognitive modelling are greatly reduced if it becomes standard practice to publish source code. However, this requires *all* source code, including that for the models, the experimental situations, the analysis, and the comparison to human data. Achieving this requires careful additions to the standard publishing process.

Chapter 6 collects these various ideas together and presents a generic methodology for computational cognitive modelling. This offers particular steps to go through, plus particular suggestions on how to adjust models, what aspects to examine, and when to add or remove measures from consideration. To support this methodology, a software toolkit (CCMSuite) is created which handles the non-standard aspects of this methodology. This toolkit is used for the remainder of the paper to examine various different modelling situations.

Chapters 7-13 apply the methodology and the toolkit to the repeated binary choice task, the formation of peer groups in children, and a model of ants learning to use pheromones to forage. These represent the wide range of modelling situations that appear in cognitive science, and are meant to demonstrate and validate the various aspects of the modelling methodology. For each situation, the methodology produces novel conclusions, providing useful information about the models which would not be available using the standard approaches currently used within cognitive science.

2 Computational Cognitive Modelling

Before developing an overall methodology for performing computational cognitive modelling, it is necessary to first take a closer look at what exactly is meant by the term, and what the scientific intent is that motivates the development of such models.

2.1 Cognitive Modelling

Cognitive science involves the study of the most complex system in any science. To overcome this difficulty, cognitive scientists have had to carefully examine traditional scientific techniques so as to determine how to best apply them to this domain. One of the techniques that has been widely used within cognitive science is the idea of cognitive modelling. This is the idea that in order to understand the mind, one needs to develop models of the mind. These models are meant to be explanations of observed cognitive behaviour. By developing more complex and more accurate models, it is possible to explain and predict more aspects of the intelligent behaviour of living organisms.

2.1.1 Origins: Beyond Behaviourism

The creation of cognitive models can be seen as the key element which distinguishes cognitive science from earlier approaches to the study of the mind, such as behaviourism. In this earlier approach, theories about the mind were only meant to refer to stimuli and responses, not anything that might happen in between. After all, if one cannot observe what is happening within the mind and brain of the organism, why make it part of one's theories?

“In a reaction against the subjectivity of armchair introspectionism, behaviorism had declared that it was just as illicit to theorize about what went on in the head of the organism to generate its behavior as to theorize about what went on in its mind. Only observables were to be the subject matter of psychology.” (Harnad, 1990).

The underlying idea here is that it is not technically necessary to postulate any internal aspect to cognition. Any theory that states that some stimulus causes some internal processing in the brain that then causes some overt behaviour can be replaced with one that goes directly from stimulus to response. In other words, theories can eliminate the unnecessary middle step of investigating what exactly goes on inside the brain, and can instead focus on finding the overt relationships between what is sensed and the resulting actions. This approach worked very well for the first part of the twentieth century, resulting in such well-known rules as classical and operant conditioning.

However, eventually this methodology of simply relating stimuli to responses was found to be inadequate. In particular, the relationships between stimuli and responses become more and more complex, frustrating the hopes of those looking for a direct mapping between senses and actions. In the classic water maze experiments, for example, rats who learn to navigate a dry maze by running can clearly still make use of this knowledge when the maze is flooded and they must swim, even though the stimuli and the responses are considerably different. Developing an explanation of this behaviour must involve some discussion of what is “going on” in its head.

This leaves cognitive scientists in a difficult situation, in that the only data points available are the stimuli and the responses. These are the only “observables”. It is thus necessary to develop explanations that explain a process whereby these stimuli lead to

various responses. Any explanation more complex than those of the behaviourists (who focused on direct mappings between stimuli and responses) is going to involve a description of internal aspects of the brain responsible for the observed behaviour. These descriptions are what are referred to as *cognitive models*.

2.2 Computational Modelling

A computational model is a specific sort of model: one that has been specified in a manner suitable for running as a computer program. At first, this may seem like an unnecessary distinction, as surely the mere fact that a model has been written as a computer program rather than as a text-based or diagrammatic description should not fundamentally change the scientific use of the model. However, the creation of a computational model leads to two important results: the complete specification of that model, and the elimination of predictive ambiguity.¹

2.2.1 Black-Box Models

By implementing a model as a computer program, the result is a *fully specified model*. This should be distinguished from models which describe the interactions between various components but which leave the internals of the components unspecified. These are models that describe how the system as a whole works, but not how the individual components work. For this reason, we can call them *black-box* models (meaning that the internals are hidden from view). A key limitation of these black-box models is that they

¹ It is, of course, possible for a non-computational model to be fully specified and non-ambiguous, given a sufficiently detailed description and mathematical proofs of behaviour. However, it is the view of this author that such detail is rare in practice, and when such detail is present, the resulting model could then be thought of as equivalent to a computational model.

allow for *qualitative* predictions, but not (in general) *quantitative* predictions. Without knowing how the individual parts work, it cannot be determined how quickly the system as a whole will work, or how accurate it will be, or exactly what responses it will give. Instead, researchers must speculate that modifying the functioning of one component in a certain way might positively or negatively affect the overall performance, based on how that component interacts with the other components.

For example, consider the black-box model presented shown in Figure 2.1. This model (from Shallice & Burgess, 1996) depicts the authors' theory as to the processes involved in determining a strategy (or schema) to follow in novel situations. There are three separate stages, each of which contains various processes which act in parallel but affect each other via the connections shown. Each of these processes are described in the original paper by giving a brief verbal description. For example, the Delayed Intention Realization Marker Process is for “the formation and realization of intentions so that one can prepare a strategy and plan action for a later time” (ibid, p.1406).

This sort of black-box model does make a number of empirically testable scientific claims. Impairments to processes which connect to other processes should adversely affect those other processes. Tasks which only involve some processes should not be affected by damage to processes not involved. It may even be possible to localize these processes to various regions of the brain by examining impairments in people with various brain lesions. In the case of this model, evidence such as “Of the 46 patients [with posterior lesions], 78% indicated that they developed [a particular] strategy compared to only 50% of the 46 patients whose lesions involved the frontal lobes” led to

the conclusion that “of the four processes to be considered, two are impaired by frontal lobe lesions” (ibid, p.1407).

The key limitation to this sort of model can be seen in the sort of evidence used in the above example. The fact that the numbers are 78% in one case and 50% in another are not important: all that is important for the purposes of this black-box model are that there is a statistically significant difference between these values. Information about the quantitative size of that difference is not something that is expected from such a model. If a model is to do more, then it needs to be specified in more complete detail. If it were to be specified to such a degree that it could be implemented as a computer program, then it would be possible to form precise numerical predictions as to what would occur in various situations. Of course, such detail may not yet be appropriate or possible for a model as complex and as high-level as this one.

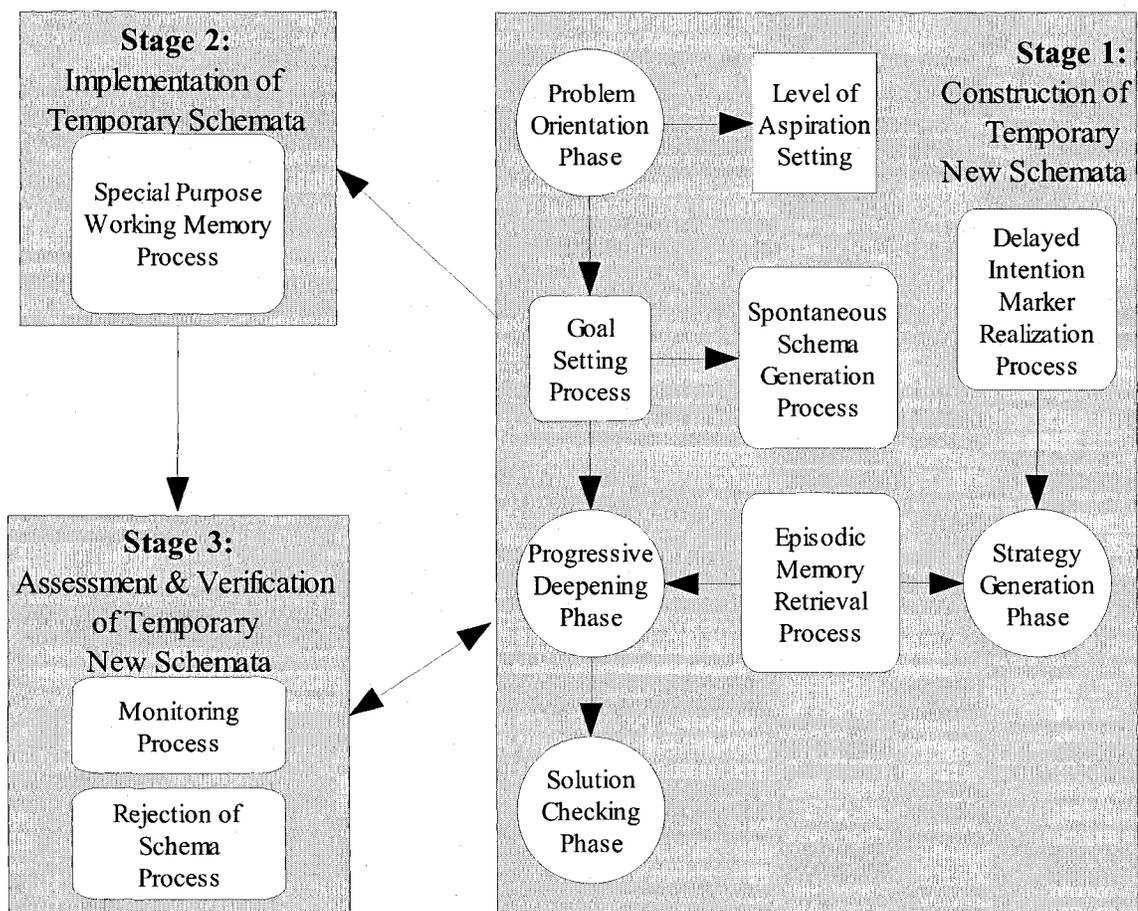


Figure 2.1: A black-box model of high-level decision making. Adapted from (Shallice & Burgess, 1996, Figure 1)

As will be discussed further in section 4.1, it is possible to see the use of computer programs as having the same role in cognitive science that the use of mathematics had in the development of physics. Much of early physics had exactly the sort of qualitative predictions that are found with black-box modelling. The creation of the mathematical formulation, however, immediately led to powerfully exact quantitative predictions. It is this transition that is the promise of computational modelling. In particular, it allows there to be completely non-ambiguous predictions from a model, as opposed to more

qualitative boxes-and-arrows models, which require an undefined step to interpret how differing situations will affect model performance.

2.2.2 Complete Specification

The alternative to the black-box approach is to give an explanation as to how each of the components within a model functions. Each of these components can be broken down into smaller components, which can be further broken down. To avoid infinite regress, a set of primitive components is needed that all others will be built from. For a computational model, this set of primitive components is the programming language the model is written in (i.e. the collection of elementary operations and data structures which comprise that language). Thus, every component in a computational model is a *fully specified computer program*. One question to keep in mind is exactly how this program, and in particular its inputs and outputs, can correspond to the real world situation.

Certain aspects of this model may be the same as that seen in the real creature (the number of errors it makes, for example), but how it expresses that behaviour is extremely different, as it will generally exhibit behaviour by outputting numerical values that are saved into a data file, instead of physically pushing buttons or verbally speaking results. This issue will be directly addressed in section 2.2.4, below.

An important effect of this complete specification is that *all* of the assumptions required by the model must be specified, “including aspects which might otherwise be overlooked” (Cooper et al, 1996). There are thus no hidden assumptions in the chain of logic which leads to the predictions: all of the assumptions are built into the code of the

computer model itself. The necessity of this has long been known within cognitive psychology:

One of the chief benefits of the model-based research approaches is the requirement they impose on the researcher to be explicit about assumptions concerning the factors that are involved in performance and the ways in which these factors combine to produce the observed responses. Once such assumptions are out in the open, they can be evaluated for psychological plausibility or put to empirical test. When the assumptions concerning the (possibly multiple) causes underlying test performance are not explicit, Group x Task interactions may mislead the researcher about the foundations for her process-related inferences (Cole & Means, 1981, 135).

Of course, given the complexities involved in creating a computer model, these assumptions may not necessarily be explicit *to the modeller*. It is always possible for that individual to create a model that makes certain assumptions without knowing that one has done so. This is especially true if a modeller is making use of software written by someone else, so that the model is built upon a computer program that contains aspects that the modeller is not aware of. However, even in this case, all of the assumptions are, in principle, available to anyone who wishes to delve into the computational implementation. This allows other researchers to accurately evaluate modelling work.

Computational models also have the interesting feature that they can produce the *same sort of data* as the real subjects. That is, the same statistical analysis can be performed on both the raw observed data and on the raw data from the model. This greatly simplifies the task of ensuring that the comparison between model and reality is done appropriately.

2.2.3 Models as Hypotheses

It is important to note that computational models can be considered to be scientific

hypotheses. That is, a particular model is a hypothesis that this particular computer program describes some particular real cognitive behaviour. However, it is usually necessary to actually *run* the model in order to determine what the model predicts.² In other words, the model is generally *opaque* in terms of its predictions: one cannot simply look at the model and know what it implies.

This situation differs considerably from what has become common practice in cognitive psychology. In that domain, the hypothesis is often *identical* to the prediction itself. For example, one might hypothesize that changing the colour of text will increase the time needed to read it, or that people who are given alcohol will perform less well on motor-skills tasks than those who were not. Here, there is no difference between the hypothesis and the prediction. In these simpler situations, scientists rely on the fact that if the prediction is true, then the hypothesis is true, and if the prediction is false, then the hypothesis is false.

In the case of computational modelling, the hypothesis (H) implies a prediction (P), and the truth of this prediction can be experimentally confirmed. So, we have $H \rightarrow P$, and if it is determined that P is false, then it can be concluded that H is false. However, if P is true (i.e. if the model is accurately predictive), then the conclusion is more ambiguous.

It is clearly not correct to simply conclude, on the basis of P being true, that H is true (i.e. that the model is correct). More precisely, one cannot conclude that the model is the (only) accurate method for producing predictions. This is because $H \rightarrow P$ does not mean

² In some situations, models may be amenable to analytic tools which can produce results without running the model. However, these tools are not usable on every cognitive model, and the methodology developed in this dissertation is meant to be generally applicable to any modelling work.

that $\sim H \rightarrow \sim P$. Finding one predictive model does not mean that other models are going to be less predictive. Certainly, it is a good first step, but it is quite possible that other (perhaps very different) models could also give a sufficiently similar result.

This also means that it is not sufficient to follow the typical approach of testing a single hypothesis at a time. Instead, a multiple-models approach is required, resulting in $H_1 \rightarrow P_1$, $H_2 \rightarrow P_2$, $H_3 \rightarrow P_3$, and so on. This allows for the identification of one model (or one group of models) which gives sufficiently accurate predictions, while at the same time noting that other models do not. Without such evidence, one cannot be sure that there is something important about the structure of a particular model that produces the accurate predictions. After all, it could be that *any* reasonable model might produce accurate data. It is incumbent on the modeller to disprove this possibility.

Most importantly, this calls into question the increasingly common approach of deciding upon one hypothesis that is thought to be the 'right' one, and running an experiment (or simulation) to confirm or deny this hypothesis. Experimental methodology is meant to decide *between* competing hypotheses, and is not a situation where one should be *hoping* for one particular outcome (thus introducing potential biases). This is particularly a problem in computational modelling, where it is common for researchers to have particular modelling architectures that they favour.

Furthermore, since it is clearly impossible to compare models against *all* possible other models, modelling must be performed in such a way that it is *easy for other researchers to do the exact same test using new models*. It is both highly likely and scientifically

desirable that future researchers will want to continue to explore a given situation with new models, or make slight changes to the experimental situation to attempt to tease apart the results of currently equivalent models. One of the great advantages of computational models is that this level of sharing is possible, although as will be seen in chapter 5, this does not always occur.

2.2.4 Modelling the Environment

In order for a computational model to generate a prediction about a subject's behaviour in a particular experimental situations, *a model of this situation is also needed*. In general, *any* computational cognitive model must be accompanied by some sort of environment for it to interact with. If this model is similar to the environment of the experimental situation, then the performance of the model within that environment can be compared to that of the human subjects within the real environment.

Indeed, the process of defining a computational model of the environment (i.e. the situation in which the model is to be evaluated) provides the same sort of possibility for precision as that of creating a computational cognitive model. Furthermore, given the classic Agent-Environment diagram shown in Figure 2.2, modelling either the agent or the world should be a similar process, given the evident symmetry. The outputs for the agent are the inputs for the environment, and vice-versa. It should be noted that throughout this thesis, the terms “experimental situation” and “environment” will be used to refer to the system that the agent is interacting with. An experimental situation will be used to refer to a particular well-defined situation, while environment is a more generic

term.

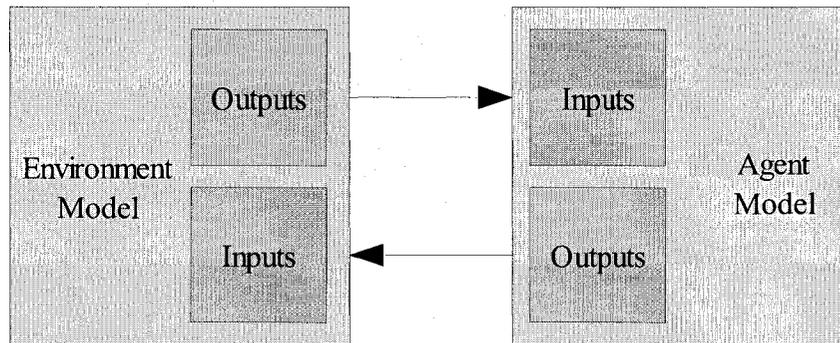


Figure 2.2: Agent-Environment interaction

Identifying the model's environment as a model itself indicates that the computational environment must be made to be *similar* to that of the real environment. However, the environment model will not be *identical* to the actual environment: instead of a light turning on to alert the experimental subject, in the computational model a particular value may be changed from 0 to 1. The important criterion here is that there are aspects of the computational environment that take on the same *functional role* as the corresponding aspects in the real environment.

Interestingly, the importance of this correspondence can be seen to be similar to a well-known problem in cognitive psychology. In *Comparative Studies of How People Think* (Cole & Means, 1981), a variety of methods are presented for comparing the performance of subjects *from different groups*. This is encountered in any situation where one cannot randomly assign subjects to an experimental and a control condition; for example, comparing the reaction times of people of different ages, or comparing the reading ability of people raised in different cultures. In particular, they identify two types of problems

that occur when “all other things are not equal:”

(1) Our comparison groups will generally differ on not one but a multitude of dimensions. Hence, we will have difficulty in proving that any one particular difference between our groups is the source of the observed differences in performance. (2) We will not be able to ensure that our treatment is equivalent for different groups at all points in the experimental task. If our groups are, in a psychological sense, receiving different treatments, the interpretation of observed group differences is further clouded. (Cole & Means, 1981, 35)

If these problems are translated into a modelling situation, then the first type deals with the comparison between the model and the subjects. It is certainly clear the a model differs in many ways from the real-world subject being modelled. Furthermore, *any* of these differences could potentially give rise to differences in behaviour. Exploring these differences is the entire point of cognitive modelling. However, the second type of problem identified by Cole and Means is specific to the *treatment* (i.e. the *environments* of the model and the real subjects).

When comparing two different human populations, it is certainly likely that one may have to have different stimuli. For example, when using a questionnaire to compare English and Chinese populations, *the questions must be translated*. That is, the appropriate stimulus is different for the two groups. The same situation will happen when comparing models to real organisms.³ As Cole and Means point out, “there is reason to suspect the most standard experimental procedures are neither equally motivating for, nor equally well understood by, college students, preschoolers, retarded adolescents, schizophrenics, and Australian tribesmen” (Cole and Means, 1981, 44).

³ Or, at least, this will be the case until computational models are developed that are complete enough to read the questionnaires and fill them out for themselves.

In one sense, this is an unsolvable problem. How can the experimental stimuli be modified so that the computer model receives it in a manner *equivalent* to the real organism? Does this mean that any comparison is hopeless until we have a complete cognitive model of everything?

Such a conclusion would doom any initial attempts at modelling. Instead, a more productive approach is to be more explicit about this comparison. To a certain degree, choosing how this comparison is made is as important as the model itself, but it tends to get significantly less attention. Cole and Means specifically point out that this comparison can go awry in any and all aspects of an experiment, and provide three general areas for consideration: 1) the task materials; 2) the provided instructions; and 3) the actions requested.

The task materials are the clearest of the points of comparison. The particulars of the experimental stimuli (words on flashcards, coloured lights, visual patterns, and so on) are going to be very different for humans and for computational models. Common analogues include using a particular input in the computer model to correspond to a particular letter in the human stimuli. Even in models of mammalian vision it is common to give the visual input as a grid of inputs representing a static picture, rather than the quickly-changing array of data generated by the human eye jumping in sudden saccades all around a scene. This is not, per se, a problem, as long *the fact it is being done is made explicit*. When the object recognition problem is avoided by giving models simple, elemental stimuli, then the emphasis is on modelling *other aspects of the behaviour*. Of course, this may not be a successful idea, as it is quite possible that by simplifying the

problem in this way, it may actually be *harder* to create accurate models. But, some simplification is always necessary, as long as these assumptions that are being made by the choice of inputs (and outputs) are made clear when interpreting the model.

Turning to the task instructions, it is clear that with adult human subjects, it is relatively easy (although not trivial) to create verbal or written instructions in their native language about what they are to do. With infants or animals, other, well-established ‘motivations’ are used to set up situations where they will (hopefully) choose to do what is desired of them (although this is also a potential source of controversy). With computational models, the situation is different. A model is often set up explicitly to do one particular task. This, on its own, is reasonably uncontroversial, as it can be argued that what is being modelled is the activity of the agent once it has understood the instructions and decided to do what has been asked. However, when a model gets used for more than one situation (as successful models will have to do), a method is needed to *tell* the model that its situation has changed. Sometimes, this is done by having certain parts of the model be inactive at some times, and then the experimenter activates them back on when required. For example, it is common with neural network models to have the learning mechanism ‘turned on’ at certain times and ‘turned off’ at other times. In such situations, it needs to be pointed out that this adjustment *is not part of what is being modelled*.

Finally, there is the procedure itself: the thing the model is being asked to do. In comparisons between human groups, it is argued that this is important because different subject pools may be differently familiar with the task in question. For example, Western school-children are unsurprisingly better at tracing a pattern with pencil and paper than

Zambian school-children. But, since this performance difference disappears in a task such as copying a pattern using modelling clay, we should conclude that Western children are more familiar with the tracing task than Zambians children, and we should not conclude that there is some internal cognitive ability which makes the Westerners better at tracing. It is difficult to know how to interpret this type of difference in the situation of computational modelling. One way is to note that computational models tend to be extremely specific to particular sorts of situations; none is as flexible and general as the human mind. From this perspective, our models are *much more familiar* with the task at hand than the real subjects are. After all, all the models ever 'experiences' is the performance of the task. On the other hand, the living subjects can apply a vast array of previous knowledge of other, possibly relation situations to the task at hand. In this sense, they are the more experienced.

Overall, then, the conclusion is that an explicit depiction of the model of the environment must be included in the presentation and discussion of any model. This helps clarify exactly what is and is not being modelled, and helps point out key differences that may be responsible for some modelling discrepancies. By making them explicit, these assumptions and modelling decisions can be questioned, leading eventually to improved and expanded models.

2.3 A Computational Cognitive Model

Numerous computational models already exist within cognitive science. This section describes the RELACS model (Reinforcement Learning Among Cognitive Strategies),

developed by Ido Erev and colleagues (Roth & Erev, 1995; Erev et al, 1999), and discussed most recently in a Psychological Review article (Erev & Barron, 2005). This model should be seen as a representative example of current computational cognitive modelling practice.

2.3.1 Defining the Environment

The domain of cognition that RELACS is meant to model is *repeated binary choice* behaviour. These are situations where human subjects have to make a series of either/or decisions, usually represented by pressing one of two buttons. Situations can vary in terms of what sort of feedback is given between choices, to indicate how good that choice was. This allows the subject to refine their choices. For example, in one simple condition, choice A may be correct 70% of the time, while choice B is correct the remaining 30% of the time. By giving feedback as to whether the correct choice was made, the subject will eventually choose choice A more often than B. The key measurements in this case would be how much more often A is chosen over B, and how quickly this occurs. This is evaluated by grouping the choices into blocks of 100 and determining the average number of A choices within each block.

In a more complex situation, instead of indicating whether the choice was correct or not, a particular number of points is given to the subject after pressing a button. For example, pressing button A may give 1 point half of the time and 10 points the other half of the time, while button B may always give 7 points. The feedback given to the subject is the number of points they receive. In the most complex situation, the feedback also includes

the amount of points they *would have received* if they had have pressed the other button. In either of these situations, the experimenter may vary the point distributions from the choices.

As per the discussion in section 2.2.4 on Modelling the Environment, it is useful to exactly specify the environment the model is to be put into, so that the degree of equivalence between the model environment and the real subject environment can be seen. In this case, the model has a single output: pressing one of two buttons. The input it receives is based on either a single reward value, or both a reward value and an alternate reward value (the value that would have been received if the other button had been pressed). In the particular case of experiment 1 (Erev & Barron, 2005), the reward for button A was 11 and for button B was 10. A total of 200 choices were made by each subject. The percentage of *A* choices in the first 100 choices is measured and called Pmax1 (the percentage of time the choice with the maximum average reward was chosen in the first block of 100 trials) , and the percentage of *A* choices in the second 100 trials is measured and called Pmax2. This process is depicted in Figure 2.3.

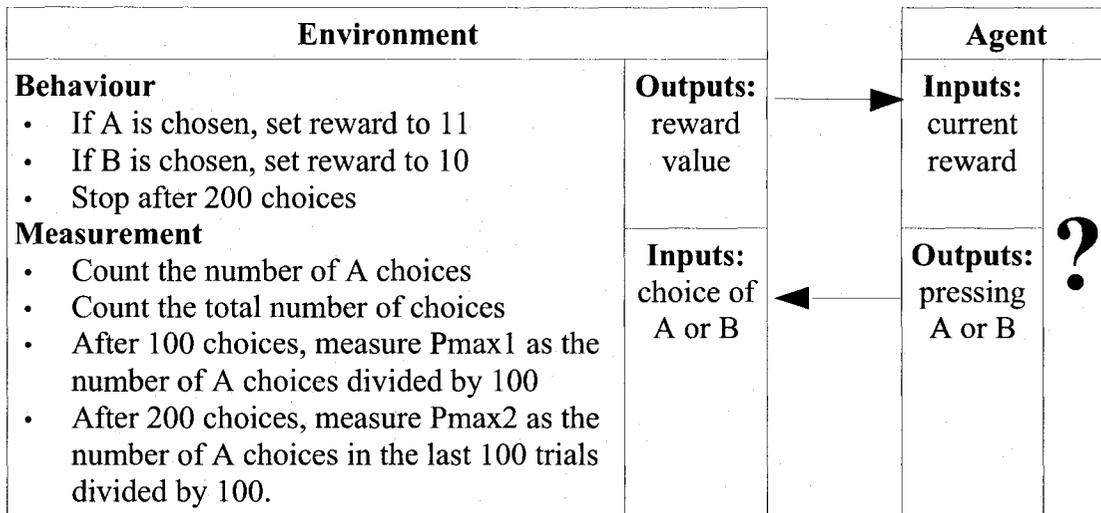


Figure 2.3: Agent-Environment interaction and environment specification for experiment 1 (Erev & Barron, 2005).

This detailed description of the environment is meant to be part-way between the prose description of the environment model and the actual computational implementation of it. It is meant to provide a non-ambiguous listing of all of the processes involved in the environment that the model will be run within, and precisely determine the particular inputs and outputs that allow the model and the environment to interact. In this case, the only input received by the agent is the reward (a numerical value) from the previous choice, and the only possible outputs from the agent model are the selection of either A or B.

Given a sufficiently congenial modelling system, it should be relatively straight-forward to convert the above environment description into an implemented computational model of the environment. While the reader is not expected to have a background in programming computational models, it may be instructive to quickly examine the actual computational model for this case. The software system used to create this model is the Carleton Cognitive Modelling Suite (Stewart, 2006), based on the Python programming

language, and is described in more detail in Appendix A.

<pre> class Environment1(ccm.Model): def start(self): self.countA=0 self.presses=0 self.reward=0 def press(self,button): if button=='A': self.reward=11 self.countA=self.countA+1 else: self.reward=10 self.presses=self.presses+1 if self.presses==100: self.Pmax1= self.countA/100.0 self.countA=0 elif self.presses==200: self.Pmax2= self.countA/100.0 self.stop() </pre>	<p>A_count is the number of times A is pressed (and starts at a value of 0)</p> <p>presses is the total number of times a button has been pressed (starting at 0)</p> <p>reward is the current reward value</p> <p>when A is pressed, set the reward to 11 and increase A_count by 1</p> <p>otherwise, set the reward to 10</p> <p>increase the number of presses</p> <p>if we've reached 100 presses, calculate Pmax1 and reset A_count to 0 to count the next 100 presses</p> <p>if we've reached 200 presses, calculate Pmax2 and stop</p>
---	---

Figure 2.4: Source code for a computational model of Environment 1 (Erev & Barron, 2005)

The intent in showing the computational model in this way is to indicate the direct correspondence between the description of the environment model (Figure 2.3) and the implementation of the environment model (Figure 2.4). This ensures there are no hidden complexities in the implementation that are not a part of the description. Having aspects of the model which are not made explicit in this way leaves open the possibility that parts of the implemented model that are not described are, in fact, responsible for important aspects of the models' behaviour.

For the remainder of this thesis, the actual computational implementation of the models and their environments will *not* be given. This is for the sake of simplicity and clarity, and to allow for a broader readership. However, all of these implementations are provided in Appendix B, for those who are interested, and to serve as a precise

specification of the models described.

2.3.2 Defining the Model

The RELACS model (Erev & Barron, 2005) is meant to deal with all three of the types of feedback situations described above. Since the model is meant to explain human behaviour in these situations, it should react to different reward distributions in a manner that is comparable to that of the human subjects. Since this model is not so simple as to be amenable to mathematical proofs, its performance must be determined using iterative computational modelling.

The model consists of three different decision rules, plus an overall rule to determine which decision rule to use at a given time. These rules are presented here in a mathematical formulation. For the complete source code for this implementation, see Appendix B.

The first decision rule is to choose the action which has the highest *recent payoff* (R_j). This is calculated as follows, with v_j as the most recent known reward for performing action j , and B as a parameter:

$$R_j \leftarrow R_j(1-B) + v_j B$$

R_j is initialized to the average reward value.

The second decision rule involves choosing random previous instances to form a belief about what is going to happen in this instance. In a situation with complete information being given, a single past example is chosen at random, and the rewards seen from that

example form the beliefs about what will happen in this next case. When there is less than complete information, previous examples are chosen at random and merged together. The decision would then be to choose the action which is remembered to have the highest reward. However, this decision rule then performs a “loss consistency test”, which involves choosing a further K previous events in the same manner (where K is a parameter). If the action with the higher reward initially has more losses and higher total losses over all $K+1$ remembered events, then the decision is reversed.

The third decision rule is a probabilistic and adaptive version of the first rule. Here, we again estimate the reward (W_j in this case) for each action. The parameter a is introduced here and is constrained to be smaller than B .

$$W_j \leftarrow W_j(1-a) + v_j a$$

Next, we determine how *stochastic* we are going to be (S), which can be thought of as a measure of certainty. Here, v_1 and v_2 are the last observed rewards from performing actions 1 and 2, and v is the last reward received regardless of action.

$$S \leftarrow S(1-a) + |(v - \max(v_1, v_2))| a$$

S is initialized to the average absolute difference between the reward and the average reward value. Given current values for W_j and S , an action is chosen using the following probability calculation, which introduces the parameter L .

$$p_j = \frac{e^{\frac{w_j L}{s}}}{\sum e^{\frac{w_j L}{s}}}$$

To decide which of the above three decision rules to use, the system uses the same mechanism as decision rule 3 (modified to decide between the three choices of which decision rule to make, rather than between the two choices of which button to press), but only recalculates W_j for the decision rule actually chosen. This results in a model with a total of 4 parameters (B , K , a , and L).

2.3.2.1 What The Model Does

The mathematical description of the model given above can be rather difficult to interpret in terms of what is actually happening. Determining when the various formulas are applied, and what actually occurs to result in a final choice of which button to press is not immediately obvious. For this reason, it is useful to give a less formal verbal description of the process followed by RELACS.

There are three decision rules defined within RELACS. Each one is independently capable of indicating which of the two buttons to press. However, these three rules do not always agree (otherwise they would be different implementations of the same rule). Thus there are actually two choices occurring within the RELACS model: the choice of which decision rule to use, and then the choice of which button to press.

The first step is thus to make the initial choice of which rule to use. This is done using the third decision rule, which will be described below. For now, it is sufficient to note

that, overall, whichever decision rule leads to better decision (i.e. more points being awarded) will be chosen more often. If a particular rule turns out to not give very good decisions, then it will be chosen less often. Initially, when no decisions have been made yet, there will be a random choice as to which decision rule to use.

If the first decision rule is used, then RELACS follows a very simple process. It keeps track of two values: the expected reward for pressing button A (R_A) and the expected reward for pressing button B (R_B). Whichever one of these is higher is the one that it chooses. However, since there is no way of knowing what the overall expected reward is for these two different buttons, it needs to estimate these values. It does this by first setting them to some default value (in this case, to the average expected overall reward). Then, each time it gets a reward, it updates those values by averaging the observed reward with the internally stored value.

For example, Table 2.1 shows how these two values change as rewards are received. In this particular case, the values are updated by the simple average of the currently expected reward and the received reward. This corresponds to a parameter value of 0.5 for B . With a smaller value, the system will weight the new value more heavily, and with a larger value there will be less weight given to it. This affects how quickly the system will switch from its expectations given new information. If one button suddenly starts given very low rewards, but in the past has given high rewards, then this parameter B will affect how long it continues to press that button.

Event	Reward	R_A	R_B
Initially (no experience)	10.5		10.5
Pressed A	11	$(10.5+11)/2$	10.5
Pressed B	10	10.75	$(10.5+10)/2$
Pressed A	11	$(10.75+11)/2$	10.25
Pressed A	11	$(10.825+11)/2$	10.25
Pressed A	11	$(10.9125+11)/2$	10.25

Table 2.1: How the first rule's internal expectations of the reward for pushing the two buttons change over time given rewards.

If, however, the second rule is chosen, then a very different process occurs. Instead of maintaining a single expected value, this rule makes use of previous events. It keeps track of each result of pressing a particular button, and memorizes the reward that occurred.⁴ When it comes time to use this rule to choose a button, it recalls, at random, a previous event where each button is pressed. In this case, after some experience it would recall pressing button A and getting a reward of 11, and then it would recall pressing button B and getting a reward of 10. These two values are then compared, and the one with the largest reward is chosen. This would lead to deciding to press button A.

The final option is the third decision rule. This starts in the same way as the first decision rule (although with a parameter a instead of B). However, instead of just choosing the option with the highest expected reward, this rule *will sometimes chose the other one*.

How often this occurs is based on how much the rewards are varying and how different the two expected reward values are. If there is a lot of variation, and if the two expected rewards are very close, then it will choose randomly between them, even if one of the values has a slightly higher expected reward. If there is very little variation, or if these

⁴ This is not meant to imply that the person has explicit access to the memory of such events.

values are very far apart, then it will be more likely to choose the one with the higher reward. Exactly how this occurs is governed by the parameter L , with large values making the system more likely to choose the best model, and lower values more likely to give the other choice a try.

As mentioned above, this third decision is also used to decide between the three decision rules. It is used in exactly the same way, but with three values to be kept track of (the expected rewards when using decision rule 1, 2, and 3) instead of just two (the expected rewards for pressing A and B). The result is a system which only outputs one particular choice (the button to press as indicated by the chosen decision rule).

This whole system is summarized in Figure 2.5.

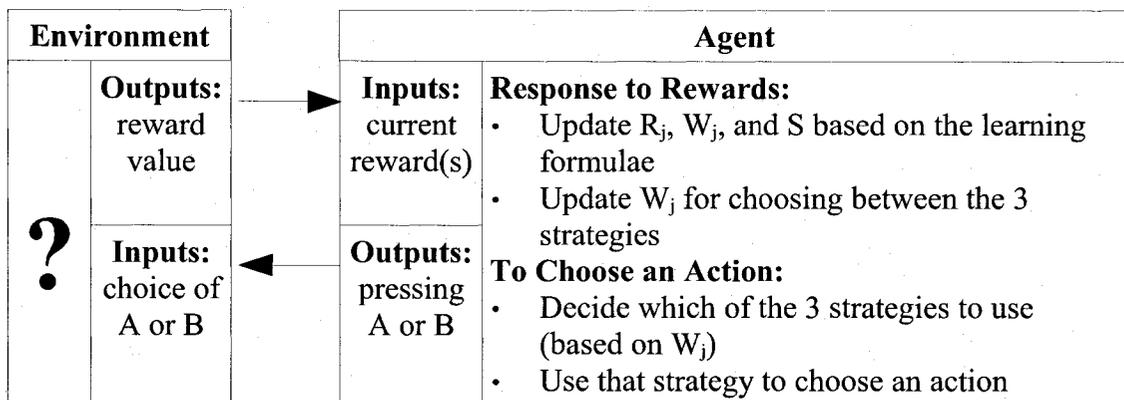


Figure 2.5: Overview of the RELACS model of repeated binary choice.

2.3.2.2 Possible Variations

Given either the mathematical definition of the model or the less formal description, it is very difficult to judge how it will perform. Erev and Barron do provide discussion about their inspirations for the particular components of the model. The first decision rule is meant to be a *Fast Best Reply*, which is meant to give a simple calculation of which

choice has provided the best results in the past, on average. The second rule is supposed to represent *Loss Aversion*, biasing choices away from ones that have caused unexpected losses in the past. The third rule is supposed to follow the principle of *Diminishing Random Choice*, the gradual change from an exploration of the effects of different choices to eventually just sticking with the option that seems best.

However, each of these possibilities could be implemented in a multitude of different ways. The formulas for calculating R_j and W_j could be replaced by any algorithm which takes a sequence of values (the rewards for that choice in the past) and combines them into a single representative number. The formulas above could be replaced by a simple calculation of the mean reward, or the median reward. These could also be windowed means and medians, so as to only consider the last few observations of the reward. More complex algorithms are also possible, such as using the ACT-R PG-C learning rule (Anderson & Lebiere, 1998), or any Reinforcement Learning scheme (Sutton & Barto, 1998).

Any of these mechanisms could perform this role in the RELACS algorithm. Each one, however, would perform it in a slightly different way, producing different results. It may be that the behavioural differences between these possibilities are extremely small, or, they may result in very different performance. Even if two algorithms do lead to identical (or indistinguishable) overall behaviour, it is desirable to find a model that best represents how humans *actually* perform this sort of task. Furthermore, these different algorithms may provide different possibilities when it comes to to determine the structure of how these algorithms are implemented in the brain. One of the requirements of

cognitive modelling research is to address this situation.

Even within a single algorithm, adjusting the particular parameters used by the algorithm will significantly change the behaviour. For example, in the case of the first learning rule, if the parameter a is very large, the algorithm will give R_j values that are based much more heavily on the results of recent choices, rather than on the initial times making those choices. How this will affect the overall behaviour is not clear, making it difficult to imagine the behaviour of the algorithm in general, as opposed to with a particular parameter setting. Indeed, it may be that one should treat each of the possible different parameter settings as a separate and distinct algorithm to consider.

In the case of the RELACS model, there is no provided justification for these algorithmic choices, or than the high-level intent for the three rules to somewhat correspond to three observed effects of human decision making. Given the complexity of the model (which is still somewhat simpler than many computational cognitive models), it is impossible to judge the how these particular algorithmic choices will affect the behaviour of the model simply by looking at the algorithms themselves. Instead, the model must be implemented into a computer program and its behaviour must be observed and evaluated.

2.3.3 Evaluating the Model

To evaluate this model, Erev and Barron (2005) present 40 different experimental conditions, including data gathered by two other research groups (Siegel & Goldstein, 1959; Myers et al, 1963). For each situation they graphically compare the performance of the model with that of the human subjects. For example, Figure 2.6 shows the Pmax

values (the percentage of time the subject chooses the option which does have the maximum overall reward) in the first and second set of 100 trials, for both the human subjects (on the left) and the model (on the right).

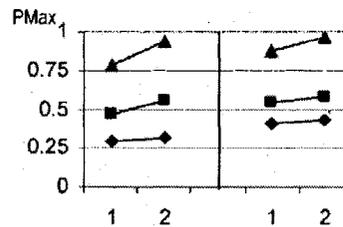


Figure 2.6: Comparison of human (left) and model (right) performance in three different conditions of a repeated binary choice task. P_{max1} is the percentage of time the reward maximizing choice was made in the first 100 trials and P_{max2} is the same measure in the second 100 trials. Image from (Erev & Barron, 2005).

To attain an overall comparison value for all of the 40 experimental situations, they then calculate the *Mean Squared Deviation* (MSD) between the model and the human subjects for all of the P_{max} values, with weighting adjusted so that each of the 40 situations had the same weighting, regardless of the number of trials involved (200 to 500).

Since the model has four parameters (B , K , a , and L), the model was adjusted to try different parameter settings to identify the *best fit* model. No details on this process are available other than “The four parameters were fitted by considering a wide set of possible values (on the space $0 < a < 1$, $0 < L < 20$, $a < B < 1$, and $1 < K < 20$)” (Erev & Barron, 2005, p. 922). The best fit that was found was $L=8$, $a=0.00125$, $B=0.2$, and $K=4$, resulting in a mean squared deviation of 0.0036. In other words, with this set of parameter settings, the average difference between the model and the human subjects' P_{max} values is around 0.06.

After fitting on these values, they then applied the model, without adjusting the parameters, to a set of 27 situations that were slightly different than the 40 original ones. Here, the MSD was 0.0040. A further set of 10 randomly generated situations were also examined, giving an MSD of 0.0053. These comparisons are meant to show that the model generalizes to new situations other than those used to determine the parameter settings.

As a further test, they compared their model to a set of 19 other variants of the RELACS model, each of which was shown to have a larger MSD than RELACS. The closest match was a version of RELACS where there was a random choice between the three strategies comprising RELACS, rather than a separate learning rule to determine which of the three strategies to use. Here, the MSDs for the three sets of data were 0.0036, 0.0042, and 0.0053, respectively. This slight difference may indicate that the aspect of RELACS that uses reinforcement learning to choose between strategies may not be an important part of the model (making it unfortunate that this aspect is what gives the model its name).

This is a typical example of how computational models are currently evaluated in cognitive science. Many questions can be raised about this process, and indeed discussing and resolving these questions is what the bulk of this thesis is about. For now, it is sufficient to simply raise a number of these issues: Is the Mean Squared Difference a useful comparison tool? How does the search for parameter values affect the model? Does this model explain the cognitive process behind repeated binary choice tasks, or does it merely predict behaviour?

These are the questions that should be resolved, or at least clarified, by this thesis. In section 4.2 it will be shown that Mean Squared Difference measures hide important details about how the model performs in various situations, and that there are alternative measures that describe the overall equivalence between the model and reality. Section 4.3 discusses the issue of the exploration of a space of parameters, showing that a wide variety of parameters can give plausible models, so the particular values given by Erev and Barron may not be the best way of analyzing the model. In particular section 4.3.1 discusses how explanation and prediction can be distinguished in these situations, and why the search for explanation requires the investigation of a much wider range of situations than mere prediction.

2.4 But Is It Really Cognition?

What is under consideration in this thesis is how cognition can be successfully *modelled*. This is a separate issue from determining whether the resulting model is *itself* a thinking, cognizing agent. There has been extensive discussion within cognitive science as to how we should interpret an accurate, working cognitive model that, for example, is able to recognize faces, learn patterns, react to pain, read and write poetry, discuss politics, and so on, and does so in a manner that is similar (in various ways) to how humans perform such tasks. Should we consider such a model to be *really* performing those tasks? Does a program that reacts to pain as we do actually feel pain? Does a program that answers questions actually understand?

These questions will not be addressed here, as these are independent from the question of

how we actually develop these models in the first place, which is the topic of this thesis.

However, I believe this is best addressed by Dennett (on the particular topic of pain, but the conclusion can be applied to any modelling topic):

[A]ny robot instantiation of any theory of pain will be vulnerable to powerful objections that appeal to well-entrenched intuitions about the nature of pain, but reliance on such skeptical arguments would be short-sighted, for the inability of a robot model to satisfy all our intuitive demands may be due not to any irredeemable mysteriousness about the phenomenon of pain, but to irredeemable incoherency in our ordinary concept of pain. Physiological perplexities may defy the best efforts of theoreticians, of course, but philosophical considerations are irrelevant to the probability of that. If and when a good physiological sub-personal theory of pain is developed, a robot could in principle be constructed to instantiate it. Such advances in science would probably bring in their train wide-scale changes in what we found intuitive about pain, so that the charge that our robot only suffered what we artificially called pain would lose its persuasiveness. In the meantime (if there were a cultural lag) thoughtful people would refrain from kicking such a robot. (Dennett, 1978)

Whether one believes cognitive modelling involves the creation of simulations of cognition or actual cognizing agents, the task is still the same: developing accurate explanatory and predictive computational models of the mind.

2.5 Summary

2.5.1 State of the Art

- Quantitative predictions require a theory to be specified to a degree sufficient to allow a computational model to be created.
- Computational models force all details of a theory to be explicitly addressed, and can identify hidden underlying assumptions in a less formal description.

- When evaluating a cognitive model, a model of the experimental situation must also be created. This should match closely to the real situation.

2.5.2 Background

- When a model is treated as a scientific hypothesis, the actual predictions of the model can be evaluated. However, if those predictions are correct, this does not mean that the model is correct. Other very different models may make the same predictions.
- The RELACS model for the repeated binary choice task provides a complex set of algorithms which controls which one of two choices the agent makes based on numerical feedback from previous choices. Evaluating the model was originally done by finding the root mean square difference between the model's average behaviour in a large number of different situations and average human performance in those situations.

2.5.3 Contributions

- The principles developed in cognitive psychology for comparing subjects from different groups were found to be applicable to the problem of comparing models to reality. These principles were shown to suggest that the precise nature of the interface between model and environment must be made explicit when interpreting modelling results.

3 Developing a Methodology: Equivalence

In this chapter, the focus is on the very heart of the modelling process. In order for something to be a *model*, there must be something which it is a model *of*. For this to be the case, there must be some sort of *equivalence* between the model and that other thing. However, characterizing that equivalence is a core issue in philosophy of science, and it has a direct impact on both the interpretation of models and on identifying what features are required for a model to be part of scientific inquiry. The results from this chapter determine what is needed in a methodology for computational modelling.

Throughout this chapter, examples will be given based on the RELACS repeated forced binary choice model described in section 2.3. This model deals with situations where a participant is presented with multiple (usually hundreds) of choices of pressing one of two buttons. After each choice, some sort of feedback is given (either a correct/incorrect indication, or a numerical point value). This feedback leads individuals to adjust their choices, given the instructions to maximize their score. The RELACS system models this process by having three separate learning mechanisms adapting to different aspects of the situation, and one overall learning mechanism which chooses which of the three strategies to adopt at a given time.

This raises the question of what this model *means*. Should RELACS be seen as a claim that there are actually three separate processes within the brain which make different decisions, and a separate process which decides between them? Are these processes physically separate in the brain? In the model, these processes are built from

mathematical operations; are these operations occurring in the brain? Or is it simply that the model produces the same overt behaviour as the brain, and the details of how that occurs are irrelevant? And what should be said about the inevitable differences between the model and reality? As is shown in the rest of this chapter, there are a variety of answers to these questions, but a common thread emerges that deals with the *process* or *algorithm* used in a model. This must be constrained to limit the model to be physically possible (at the very least), and the success of the model can be seen as an indication that particular differences between model and reality *are not important*. This is the case across a wide variety of philosophy of science viewpoints.

3.1 State of the Art: Three Types of Model/Reality Equivalence

The first major issue that any approach to modelling in science must address is the nature of the relationship between the model and that aspect of reality which is being modelled. This has been the subject of extensive discussion both within cognitive science and throughout the wider scientific community.⁵ While a complete overview of this topic is well beyond the scope of this thesis, there are a number of writers who have specifically addressed this question in terms of how it affects computational modelling within cognitive science.

In particular, there are three major ways in which a model can be seen as being in some sense *equivalent* to the real-world system it is modelling. Models can be equivalent in terms of their *behaviours*, in terms of their *algorithms*, and in terms of their *structures*.

⁵ This is clearly not a new problem, and it has been pointed out that “cognitive scientists are in the Kantian position of trying to extrapolate from experimental data to the rules and representational structures which must be thought to be responsible for that data.” (Epp, 1993)

The major question is how to establish these sorts of equivalence. How can it be known that a model uses the same algorithmic process as a given real-world system? What aspects of behaviour are important for comparison? Why are these measures important? To address these questions, each type of equivalence is looked at separately in the next sections.

3.1.1 Behavioural Equivalence

The most straight-forward aspect of equivalence is that of behaviour. Here, the model's behaviours (i.e. its measured outputs) are compared to the measured behaviour of the real system. This comparison can be performed in multiple experimental situations by adjusting the environment of the participant and the environment of the model in corresponding ways.

In the RELACS model described in section 2.3, the particular behavioural measures involved are the Pmax values: the percentage of time within a block of 100 choices that the statistically optimal choice is made. Importantly, the same measure is made for both the model and for the human participants. This allows for a direct comparison between these values. Exactly how this comparison should be made is the topic of chapter 4. This same measure is made in a variety of different environmental situations (i.e. different reward distributions, different sorts of feedback), resulting in a set of 2 to 5 Pmax values for each of 40 different situations.

The importance of this form of equivalence needs little justification, as it is generally clear that a good model should, at the very least, exhibit similar behaviour as the system

it is modelling. However, it has been suggested that it is important to measure more than one *type* of behaviour at the same time. That is, instead of just measuring the Pmax values in the RELACS case, a model should also be compared in terms of such measures as the reaction time. Instead of measuring only one or two aspects of the behaviour of the model, multiple measurements can be made of a wide variety of aspects of the behaviour. If a model produces multiple accurate outputs in a this variety of situations, then the model is in some sense more convincing, or more strongly behaviourally equivalent. This is especially true if many of these aspects of behaviour are ones that were not considered when initially creating the model.

One of the clearest calls for this approach in the field of cognitive modelling comes from Simon and Wallach (1999), who note that “the question as to what qualifies as a good approximation [i.e. a good model] has only pragmatic answers.” Their pragmatic approach is to examine data from the following categories:

- *Product*: the final outcome (reading a word, solving a puzzle)
- *Intermediate Steps*: observable aspects of the cognitive process (eye movements, thinking aloud reports)
- *Time*: how long responses take (in terms of a number predicted by the model, not how long it takes a computer to run the model)
- *Error*: both humans and the model should differ in the same ways from the imaginary ‘ideal’ behaviour (both number and type of error)
- *Context Dependency*: the effects of changing the environment or other interfering factors
- *Learning*: the effects of practice (and forgetting)
- *Lesioning*: effects of damage to the systems

They argue that models which are capable of producing accurate input/output relations in as many of these different categories as possible should be preferred. By focusing on these different *sorts* of measures (as opposed to merely looking at many different measures of the same type, but in differing situations), the model can be seen as capturing important regularities about the system as a whole.

In the case of RELACS, this means that the model should be examined in terms of not only multiple situations (as was done by its developers), but also in terms of different aspects of behaviour. This could include reaction time for making choices, and the rate of learning about those choices.

In more complex situations, the details of the errors can also be important. For example, if a model is of an arithmetic task, the fact that the model produces similar *errors* as are found in human participants lends significant weight to the idea that the model is, in some sense, *correct*. ACT-R models of arithmetic, geometry, and programming tasks exist that make errors that are highly similar to those seen in real students (for example, Corbett et al, 1993). By adjusting these models to more exactly match the errors made by one particular student, an intelligent tutoring system has been created which identifies the underlying process in the model that is responsible for the error exhibited by the human participant. By then tutoring the person on that aspect, significant learning progress has been shown.

This sort of reasoning starts to lead in the direction of *Algorithmic Equivalence*. By examining aspects of the behaviour of the model, the intent is to show that the model

produces this behaviour *in the same way* as the real system. These models are meant to be ones that follow the same underlying procedures as the real system. One way of approaching this is to simply assume that the more different aspects of behavioural equivalence that are established, the more likely it is that the resulting model will be algorithmically equivalent as well. The next section discusses a different approach to establishing this equivalence.

3.1.2 Algorithmic Equivalence

In *Computation and Cognition* (Pylyshyn, 1984) and 'Computing in Cognitive Science' (Pylyshyn, 1989), an extended case is made that it is possible for computational models to be "strongly equivalent" to their real-world counterparts. Pylyshyn defines strong equivalence as a situation where a model performs the same task, in the same way, using the same functional architecture, as the real system. The important distinction here is that we are not to be satisfied with merely matching the inputs and outputs of a system (behavioural equivalence, or in Pylyshyn's terminology, "weak equivalence"). Instead, we must establish that the underlying processes are the same. For the domain of cognitive science, this means that the functional operations that are used by the model must be the same as the operations used by the real brain, and these operations must be used in the same manner. Otherwise, our models will be mere descriptions of the overall cognitive behaviour, rather than explanations of the processes whereby that behaviour arises.

The difficulty, of course, arises in establishing this correspondence. The internal

operations of a cognitive system are, unfortunately, *unobservable*. After all, any possible observation (including such things as EEG or fMRI brain scans) can be seen as a kind of output, and does not give direct access to the internal processes. As noted earlier, this is exactly the sort of problem that led behaviourists to deny any attempt to model the internal workings of the brain. How can one ever establish that a model is more than an input/output match to the real system when all we can measure are inputs and outputs? The key aspect of Pylyshyn's argument is that we can deal with this by *treating some observables specially*.

While, in a sense, all we have is behavior, not all behavior is of the same kind, from the point of view of theory construction. By imposing an independently motivated partition on a set of behaviors, and by interpreting the partitioned behaviors in different ways, we can do much better than weak equivalence. (Pylyshyn, 1984, p. 122)

The main type of behavioural observations that Pylyshyn argues can be treated as “meta-behavioral observations” (Pylyshyn, 1989, p. 37) is “complexity-equivalence”.⁶

3.1.2.1 Complexity Equivalence

With complexity equivalence, the emphasis is on how the basic operations within a model are organized into an *algorithm* that performs the task. The difficulty is that multiple algorithms can produce the same output, so how can we know which algorithm is being used by the real cognitive system? Pylyshyn's answer is that the *time* required for the algorithm to perform its steps is measurable. Reaction times can thus be considered meta-behavioural information, and allow us to distinguish between two

⁶ Pylyshyn also discusses cognitive penetrability as a meta-behavioural observation, but this aspect will not be discussed here, as it is more directly related to Pylyshyn's discussion of models of imagery, which have not yet reached the level of computational models.

algorithms which produce the same output.

Importantly, this comparison does not require a strict determination of the amount of time each basic operation takes. Instead, we simply look at the *number of operations* used in the algorithm and how that changes in relation to the particulars of the situation. For example, suppose we are modelling the process of examining a set of objects and finding the one that is a different colour than the rest. One algorithm might proceed by examining each object in turn, while another may be based on overall gestalt principles. The first algorithm would have a number of steps that is *linearly related* to the number of objects, while the second algorithm may take a *constant* number of steps independent of the number of objects. If human performance on this task does not change significantly with the number of objects, this should be treated as evidence for the second algorithm.

There is an underlying assumption of this complexity equivalence that should be kept in mind. In order for us to conclude that a cognitive task requiring a linearly growing number of steps cannot occur in a constant period of time, we must assume that the time required for each of these steps remains roughly constant. After all, it is possible that particular basic operations are performed *faster* the more times they are used, or even the more times they will be used in the near future. In the case mentioned above, it could be that we do examine each object in turn (the first algorithm), but the amount of time needed to examine each object is inversely related to the number of objects, resulting in a constant overall time. Buller (1993) argues that this assumption of relatively constant times for operations can only be certain if we assume a strong form of type physicalism (i.e. that the operations we are specifying correspond to particular identifiable physical

structures in the brain, that then must have constant time requirements). In general, however, the complexity equivalence measure will hold as long as “there is some reason to believe that the amount of (real) time it takes is proportional to (or at least a monotonically increasing function of) the number of such primitive steps of the algorithm (Pylyshyn, 1989, p. 38).” This seems to be a plausible assumption over a wide range of situations.

The overall idea, then, is that if models are constrained to produce reaction time predictions based on the operations involved in their own algorithms (as opposed to producing reaction time predictions based on any arbitrary algorithm), then these reaction times can be seen as measures which establish algorithmic equivalence. That is, by constraining models in this certain way, what was originally seen as a measure for establishing behavioural equivalence now becomes a strong argument that the model in fact performs the same operations within the same algorithm as the real system.

3.1.3 Structural Equivalence

This goal of developing strong equivalence in models appears throughout cognitive science. In a number of situations, however, a more exacting sort of equivalence is desired which is tied to the physical implementation of the cognitive system. Many researchers augment their model development by turning to the brain itself, which is (presumably) the physical instantiation of real cognition.

For example, in *Being There* (Clark, 1997), three sorts of scientific inquiry are deemed to be equally necessary for cognitive research. These are:

- 1) identifying the “gross behaviors” exhibited by an organism (with emphasis on the complex interactions with the organism's environment⁷)
 - 2) identifying the neural components underlying those behaviours and determine the physical mechanisms for their interactions
 - 3) identifying the information-processing functions of these components, giving a “gross systems-level commentary on the roles of various components”
- (Clark, 1997, p. 127)

These three topics for concurrent investigation can be seen as mapping onto three aspects of Pylyshyn's program. The “gross behaviors” are exactly the sorts of measures used by Pylyshyn to establish weak equivalence (i.e. behavioural equivalence), the components underlying those behaviours are the fundamental operations, and the information-processing description is the algorithm built up from those operations. However, Clark specifically constrains the components to be *neural* components. That is, each of the underlying operations of cognition should be understood in terms of how neurons are organized to arise at such a component, and how that component is physically connected to other such components. The idea is that it is not enough merely to be able to describe behaviours and identify the underlying algorithms; we also need to be able to identify the underlying structures which comprise the physical implementation of these algorithms. Of course, there may not be anything particularly special about the *neural* level; it may be that the protein level or the neural group level is better suited for such a description. That

⁷ While I strongly agree with Clark's emphasis on embodied cognition and emergent properties (the core topics of his book), these will not be important issues for this thesis. The methodology developed here is meant to be agnostic to this sort of research decision.

said, Clark believes that, given our current understanding of neurons, they provide the best hope for such descriptions.

3.1.3.1 Construction

While the importance of explaining how phenomena arise from the basic physical components that comprise it is a standard of reductionist science, there is an sense in which it is even more important for cognitive science. In 'If You Can't Make One, You Don't Know How It Works', Dretske presents the following point:

I am not, however, suggesting that being able to build one is sufficient for knowing how it works. Only necessary. And I do not much care about whether you can actually put one together. It is enough if you know how one is put together. But, as I said, I do not know how to make all the right qualifications. So I will not try. All I mean to suggest by my provocative title is something about the spirit of philosophical naturalism. It is motivated by a constructivist's model of understanding. It embodies something like an engineer's ideal, a designer's vision, of what it takes to really know how something works. You need a blueprint, a recipe, an instruction manual, a program. This goes for the mind as well as any other contraption. If you want to know what intelligence is, or what it takes to have a thought, you need a recipe for creating intelligence or assembling a thought (or a thinker of thoughts) out of parts you already understand. (Dretske, 1994)

The difficulty, then, is in developing models capable of exhibiting the high-level cognitive behaviour we are interested in out of simple, understandable components. The crucial question underlying this endeavour is *what should those components be made of?* Certainly, in biological brains, we work under the assumption that these components are built from neurons (which are themselves built from organelles, which are built from proteins, and so on). It is certainly true that a computational model that is grounded in neurons (or some other physical observable) is to be preferred over one that is not, all

else being equal. But it may not be *necessary* to ground models in particular physical entities (be they neurons, proteins, or atoms) for progress to be made.

This approach of building cognitive components out of neurons is to be contrasted with the weaker stipulation from Pylyshyn that “no cognitive operator is considered primitive unless it can be realized on a computer” (Pylyshyn, 1984, p. 109). This advice ensures that it is at least possible to eventually get to a mechanistic explanation, and can provide meaningful models in the meantime.

Furthermore, it is also possible that specifying a model down to the level of its physical construction actually *over-constrains* the situation. Models that do not specify such details are more generic, and are thus applicable to many more situations. Pylyshyn (1984, p. 3) gives an example of someone dialling a telephone after witnessing an accident: if the first two buttons pressed are a '9' and a '1', any reasonable model should predict that the next button pressed is likely to be a '1'. Importantly, this is true regardless of the details of the situation, and regardless of the precise physical nature of the brain of the person doing the dialling. If the situation is changed slightly (e.g. the accident is heard, not seen; or the telephone is rotary, not push-button), then the actual physical nature of what is occurring (the particular neurons firing, the particular muscles contracting) will be vastly different. However, a high-level cognitive model of the situation would be very much the same.

The idea here is that a high-level cognitive model can capture useful generalities that may be lost when focusing on physical construction details. These lower levels are the wrong

ones to worry about, if one wants to model cognition in general, as opposed to specific instances. It is a separate research task to identify these actual physical mechanisms. Models which incorporate these finer details can be used alongside higher level models, but cannot, in general, replace them. Of course, investigation of these lower levels can be of tremendous benefit in creating overall models, so they should not be ignored completely. But cognitive modelling research does not *require* these details of the physical implementations.

3.1.3.2 Localization

Certainly, the problem of finding a detailed mechanistic account of the neural underpinning of a cognitive operation has received considerable attention and has been found to be fraught with complexity. Bechtel (1993) gives a detailed account of this process, specifically noting that the standard process of decomposition and localization (identifying particular physical components responsible for particular operations) is feasible only in situations where *direct localization* is possible. Drawing on examples from biology and connectionist systems, he argues that this tends not to be the case for such complex domains as cognition. He also argues that progress in cognitive science requires identifying operations *at the appropriate level*, rather than looking for extremely low-level neural accounts (Bechtel, 2005).⁸ The result is to use information from higher-level neural sources, such as fMRI, to identify regions of the brain that carry out particular sorts of operations, and using converging evidence from multiple behaviour which use that same region to argue that they may depend on the same underlying

⁸ Bechtel advises against attempting to account for a behaviour using operations at too high a level, as this merely shifts the explanatory burden to breaking down that operation.

operations. The assumption is that, while it is unlikely we will find direct one-to-one mappings between the cognitive operations and particular neural structures which perform them, the overall organization of the brain will be amenable to some form of operational deconstruction at some higher level. It should also be noted that there are strong evolutionary reasons to believe that many of the more recent developments of the human brain are based on making new use of previously existing operations (Anderson, 2006). That is, we should expect that the neural underpinning of newer behaviours should be distributed widely throughout the brain, using many operations based in older brain regions. This will significantly complicate the task of finding direct correspondence between brain regions and cognitive functions.

3.2 Background: Interpreting Equivalence

While the discussion of the three different types of equivalence is the main topic in philosophy of science that is needed for this thesis, it is entwined with another, related issue. This is the question of how to interpret what it is that models represent. If a model has been shown to have the identical *structure, algorithms, and behaviour* of the real system, then there would be little debate that the model truly represented the real system. However, this will rarely be the case, and indeed it may be argued, as it is above, that structural equivalence is not desired if the model is meant to explain a general class of behaviours. Instead, modelling is left to rely on such indirect measures as the complexity equivalence of timing information, and a range of behavioural data such as error rates and the impact of changing situations. Even when these are shown to be equivalent in these terms, what can be said about the resulting models?

This question is one of the fundamental issues of scientific inquiry: how should the entities within scientific theories be interpreted. A complete overview of this topic is well beyond the scope of this thesis, but it is worthwhile to consider three common and disparate points of view on this question. These can be found in the works of Giere, Thagard, and Dennett.

3.2.1 Interpretation 1: Giere's "Principles"

The first approach is to simply argue that all there is in science is prediction. That is, if we have a model that is highly predictive in many different situations, and it generalizes to new situations, then that *is* an explanation, and we do not need some other form of evidence to support the utility of the model. In *Science Without Laws*, Giere (1999) presents this point about science as a whole, arguing that science does not uncover universal 'True' Laws of Nature which explain how the world works. Instead, science is the discovery of *principles* which allow for the creation of predictions about the world. Giere's argument for the use of modelling entails that there is *only* prediction: explanation is simply a term for what happens when we have a model that is highly predictive in a wide variety of situations (Giere, 1988; Giere, 1999).

Principles, I suggest, should be understood as rules devised by humans to be used in building models to represent specific aspects of the natural world. Thus Newton's principles of mechanics are to be thought of as rules for the construction of models to represent mechanical systems, from comets to pendulums. They provide a *perspective* within which to understand mechanical motions. ... What one learns about the world is not general truths about the relationship between mass, force, and acceleration, but that the motions of a vast array of real-world systems can be successfully represented by models constructed according to Newton's principles of motion. (Giere, 1999, 94-95)

While this approach may be convincing to some, it is not a mainstream position in philosophy of science. Indeed, equating prediction with explanation has long been criticized on causal grounds. For example, one can predict the height of an object from the height of its shadow, but one would not consider the shadow to *explain* the height of the object. In Giere's approach, this sort of causal direction must come from the domain of applicability of the principles being used. In this case, a theory allowing one to determine the height of an object based on its shadow height might not generalize to as many different situations as a theory which worked the other way around.

If this approach is taken, then the RELACS model is a predictive tool that allows scientists to predict what will happen in a given repeated binary choice situation.

Certainly it will not predict exact sequences of choices (i.e. exactly what a person will choose at any given point), but it should accurately indicate the percentage of time that A is chosen rather than B, and how this percentage changes over time given different forms of feedback.

As it stands, RELACS is a highly specialized model, only suitable for one particular kind of situation. Future development may lead to more generally applicable models, and RELACS may end up being seen as a special case version of these general models, much as Newtonian gravity can be seen as a special case of Einsteinian gravity. Importantly, this means that the particular internal components of the RELACS model (the three strategies and the particular stored values) may not exactly map on to real components of the real brain. However, as the model is adapted and changed to become more general, the components within the model should become more representative of the real system.

3.2.2 Interpretation 2: Thagard's "Analogies"

A different approach can be found in the analogical processes found in *Mental Leaps* (Thagard, 1995). Here, the modelling process is recast as a way of explaining a cognitive phenomena by way of an *analogy* resulting in an *inference to the best explanation*. If there are broad, systematic correspondences between a model and the real situation (i.e. if many different aspects of the system and its behaviour, match to those of the model), and if there are no alternative better analogies available, then one is justified in mapping the explanations from the model to the real system. If, in the model, a particular behaviour can be explained by the interaction of particular components, then the analogous components in the real system explain its behaviour. Since Thagard establishes that this sort of reasoning is used throughout science, cognitive modellers can also be justified in its use. Importantly, analogy is treated by Thagard as a process of *coherence*, where no one measure is a requirement, but instead analogies are evaluated by a set of "soft constraints" which combine the important aspects of the analogy: that it is able "to explain a lot, to do so simply, to be compatible with what is known, and to be itself explainable" (Thagard, 1995, p. 174). The goal is not to create analogies (models) which are ideal on all of these aspects; rather, the goal is to produce analogies that are as coherent as possible across these many factors.

Furthermore, Thagard's approach explicitly encourages the *comparison* of models. If models are used as analogies leading to the best explanation, then multiple models are needed. The quality of an analogy can only be evaluated in relation to other analogies. Thagard provides the example of Darwin's evolutionary theory supplanting a more

creationist view of the world, based on the competing analogies of human construction and genetic selection. Since the real world matches more consistently with the processes and outcomes implied by the evolutionary theory, we are justified in believing it. Before such a theory was available for consideration, the creationist account was the best explanation.

In the case of the RELACS model, acceptance of RELACS as an explanation for human behaviour is dependent on there being no other better explanation available. Comparing explanations is, of course, highly problematic, since there are a variety of factors to consider. One model may apply to more situations, but give less accurate results. Another model might be highly complicated, making it difficult to understand or make use of, but match more closely to the known underlying neural structure of the brain. These different factors make it difficult to judge the relative overall quality of the models. However, if there is a particular model which stands out among the rest as being significantly better overall, then it can be accepted, until such time as a better model is developed.

3.2.3 Interpretation 3: Dennett's "Patterns"

In *Real Patterns* (Dennett, 1991), and in other works, Dennett argues that theoretical constructs such as beliefs are exactly as *real* as such physical science constructs as centres of gravity. Following the lead of Ross (2000), this view can be generalized to a definition of what makes something *real* that is not based entirely on its reducibility to pure physics, and is not purely a matter of pragmatic utility. If this approach can be

applied to computational cognitive models, then it can be argued that the entities postulated by the models are real patterns within the system being modelled.

Dennett's approach starts with noting that some philosophers believe that centres of gravity are *real*, while others do not. This raises the question of the reality of strange abstract objects as "Dennett's lost sock center: the point defined as the center of the smallest sphere that can be inscribed around all the socks I have ever lost in my life." He says:

These abstract objects have the same metaphysical status as centers of gravity. Is Dretske a realist about them all? Should we be? I don't intend to pursue this question, for I suspect that Dretske is — and we should be — more interested in the scientific path to realism: centers of gravity are real because they are (somehow) good abstract objects. They deserve to be taken seriously, learned about, used. If we go so far as to distinguish them as *real* (contrasting them, perhaps, with those abstract objects that are *bogus*), that is because we think they serve in perspicuous representations of real forces, "natural" properties, and the like. (Dennett, 1991)

The argument provided by Dennett for treating something as *real* rests on it being both *efficient* and *predictive*. That is, a "real pattern" is something that allows for the prediction of properties⁹, and does so based on the smallest amount of information.

Dennett's purpose in presenting this argument is to justify the treatment of human "beliefs" and "desires" as real entities, but it is straight-forward to apply the argument to computational models. In doing so, Ross' (2000) interpretation of this approach is useful. The following is his definition of what it means for a pattern to be *real*.

To be is to be a real pattern, and a pattern is real if [and only if]:
(i) it is projectible under at least one physically possible perspective

⁹ These predictions are not required to be 100% accurate, but should have a predictable amount of variation.

and

(ii) it encodes information about at least one structure of events or entities S where that encoding is more efficient, in information-theoretic terms, than the bit-map encoding of S , and where for at least one of the physically possible perspectives under which the pattern is projectible, there exists an aspect of S which cannot be tracked unless the encoding is recovered from the perspective in question. (Ross, 2000, p. 161)

That is, real entities must (i) be describable in physically possible terms, and (ii) be a more compact (in an information-theoretic sense) description of some state of affairs (referred to as S) than the raw description of S 's observed behaviour. If some observed pattern in the world can be captured by such a physically possible model, and if that model is more compact than a mere listing of the descriptive behaviour of the model (i.e. behavioural equivalence), then it meets the criteria for being real. In other words,

To *exist* is, essentially, to persist as a distinguishable entity for a long enough period of time that measurement of a set of distinguishing properties is possible using some physically possible property-detector. (Ross, 2000, p. 163)

If this "Rainforest Realism" is applied to the results of computational cognitive modelling, then the models themselves indicate the required "encoding of information".

If the properties picked out by those models, both in terms of external behaviour and internal processes, are in fact best represented by this model over all others, then the model is correct and the entities it postulates do in fact exist. Of course, if a more efficient encoding (i.e. model) is found, then that should replace the current one.¹⁰

For RELACS, this means that the key criterion for preferring one model over another is the efficiency of that model. If RELACS is the best model in terms of having the most

¹⁰ Importantly, since the replacement of one model with another is based on the well-defined principles of information theory, this replacement is not a subjective judgment. There does exist one best model for any given set of properties/perspectives; the job of science in this view is to find it.

predictive power for the amount of detail in the model, then the conclusion is that RELACS accurately represents what is actually going on within the brain.

3.3 Contributions: How to Establish Equivalence

In section 3.1, a variety of methods have been presented to establish that a cognitive model is more than merely “weakly equivalent” to the system being modelled. To establish that a model not only produces the same behaviour, but also does so in the same way, we can examine the complexity of the algorithms in the model, ground the model in the physical structure of the brain, and establish the utility of the model over a wide domain of situations and measures. All of these are useful and powerful methods for validating cognitive modelling work, and to a certain extent each of these aspects will be a part of the overall modelling methodology advocated in this thesis.

In section 3.2, multiple ways of interpreting a model were presented. Models could be looked at purely for their predictive power. Alternatively, they could be evaluated by looking for a best overall model across a range of soft constraints, with the best overall model being thought of as the best explanation. As another option, they can be evaluated in terms of their efficiency, with the most compact model for a given level of detail characterizing what is “really” going on in a situation. For the purposes of this thesis, no decision is needed about which of these stances should be taken. Instead, the conclusions of this thesis should hold *regardless* of one's interpretation of models.

When these ideas are applied to the particular circumstances of computational modelling, three novel points arise. First, Pylyshyn's algorithmic complexity measure can be

expanded into a numerical time measure, while also constraining models to contain steps which move forward in time. Second, models must be amenable to having their components be modelled themselves, but do not have to provide a complete reduction of each model all the way down. Third, by explicitly including some aspects of the real situation, and excluding others, computational models can indicate exactly which aspects of the situation are important. Importantly, all of these conclusions hold regardless of the stances on the interpretation of models discussed in the previous section.

3.3.1 Time and Causality

Pylyshyn's discussion of algorithmic complexity rests on the key observation that *time* must be treated as a special sort of output from the model. We cannot expect a “reaction time” output calculation in the same manner as we would expect an indication of what button the model decides to press, or what other action it predicts the subject will make. Instead, the time required for a particular cognitive process *must be built up from the times required for the individual operations that comprise it*. If the model requires twenty sequential iterations of some particular step in situation A and fifty in situation B, then it must also predict that situation B will take more time (i.e. will have a longer reaction time). This forms a hard constraint on the temporal predictions of any computational cognitive model.

However, this is not to say that we must find a particular time required for every step in the algorithm; after all, it is perfectly possible that operations get faster the more they are used, or that other sorts of effects will happen. We must not assume that every type of

underlying operation maps nicely onto a particular type of neural structure which always follows the same temporal steps to produce an output in the same period of time.

“Because we distinguish between the operation as a computational event, and the particular physical events that carry it out on particular occasions, all occurrences of a particular operator need not (by definition) have a unique duration associated with them, just as they need not have a unique size or location in the brain” (Pylyshyn, 1989). This sort of view would be based in a rather literal view of the computational metaphor, and indeed is not even true of modern digital computers. But, as long as we are willing to accept the assumption that cognition is ultimately constrained by basic laws of physical causality, we can constrain the models so that there is an underlying arrow of time which cannot be reversed, and any cognitive operations must occur while time moves forward. While this seems like a rather obvious constraint, what this actually does is force models to be *process* models. That is, there must be some sort of internal temporal process within the model that is meant to map onto the internal processes of the real system.

This means that a cognitive model which explains a given behaviour by indicating what operations are involved must give a time prediction based on the times required for each of the operations. In ideal situations, this prediction can be a numerical prediction: an output of the model that indicates how long a process will take to occur. This can be compared to the actual time required in the same way as any other output of the model. However, in order for this to occur, we must have numerical values for the time required for each individual operation within the model. As has been pointed out, this value can certainly vary, but if we are to achieve numerical reaction time predictions (as opposed to

the complexity measure indicators examined by Pylyshyn discussed in section 3.1.2), then part of our theory must involve a time prediction for each operation.

When examining these time predictions, however, it is vital to distinguish the time predictions produced by combining the time requirements of the components of the model from the actual amount of time it takes to *run* the model. The amount of time it takes to run the model is constrained by the speed of the computer it is run on, and this is not what the timing considerations discussed here should reflect. The time predictions of a model must be completely independent of the computer itself, otherwise the results would change whenever a new computer was used. Indeed, it is possible for a computer model to indicate that billions of operations occur in a microsecond, even though it is impossible for the computer it is running on to do so.

The underlying point of this discussion is that if algorithmic equivalence is desired, computational models must be constrained in the approach taken to producing time outputs. If a cognitive modeller wants to be able to argue that a model follows the same processes/algorithms as the real system, then that model must be constructed such that the reaction times are produced based on the internal timing of various aspects of the model. This constrains the model to be a *process* model. That is, there must be an internal temporal aspect, such that operations occur over particular periods of time, and information is passed between components of the model, but only *forwards in time*. That is, the model cannot make use of information that has not yet been calculated in simulation time, even though the separation of real time and time within the model makes this possible.

Although this constraint may seem strange, it turns out that models regularly violate it. For example, the RELACS model discussed earlier requires knowledge of the *average reward* and the *average absolute difference between the reward and the average reward*. These are used to initialize the values of R and S , respectively. However, this information is not available until after significant exposure to the experimental situation, but is made use of in the model right from the very first choice being made. In other words, the RELACS model actually requires information to move *backwards in time*¹¹. While this is possible within a computer model, it does not conform to current constraints on physical possibility. As the RELACS model currently stands, it *cannot* be seen as being algorithmically equivalent to real human subjects, regardless of its actual behaviour.

3.3.2 Reductionism

At first glance, it may seem that building up reaction time outputs for an overall cognitive process from the reaction time outputs of the basic operations comprising that process would be an almost impossible task. After all, in order to produce the reaction time outputs for any of the basic operations, one can apply the same logic and have to break that operation down into its fundamental sub-operations, and those operations down into sub-sub-operations, and so on. Then, perhaps, if those sub-sub-sub-operations are grounded in actual neurons, and we know the timing information for the neurons, then we could build all the way back up to produce the overall timing information. This may be possible in some highly constrained situations with current models, but this is clearly not

¹¹ Or, equivalently, it requires access to information which is not available to the human subjects.

appropriate for the vast majority of situations where reaction times are interesting for cognitive modelling. After all, following this approach to the extreme results in an attempt to establish full structural equivalence.

The solution is to note that this problem of regressing to lower and lower levels *is not needed*. In a cognitive theory, we *can* simply indicate a time required for the operations and still have a high-level accurate, strongly equivalent model of a cognitive process. That is, we can explain the overall process *without explaining how the individual operations work*. This expands on Pylyshyn's stance that all cognitive components need to be computable (i.e. they need to have some possible implementation) by noting that operations should also indicate how much time they require as part of the theory. This time may be constant, or it may change due to various factors. These components which calculate their time requirement (as opposed to deriving this from the time requirements of its own sub-components) form the lower-level extent of the model. Future research can break down these components into sub-components, down to the neural, protein, atomic, or even quantum levels, but this does not all have to occur at once.

For example, the RELACS model makes no attempt to describe how the mathematical operations that make up its overall algorithm are physically instantiated. It may be that those operations could be grounded in neurons or large neural groups. A model that does provide these deeper explanations of the operations and sub-operations is to be preferred over one that does not, all else being equal, but this is not a strict requirement. In chemistry it is certainly believed that all of the chemical explanations could be built up out of quantum mechanical models of atoms, but this is not a particularly practical

approach in all situations, and indeed it is an unsolved problem in any complex situation.

A similar situation is likely to happen for cognitive science.

Furthermore, it is worth noting that the majority of “neural” models encountered in cognitive modelling are not in fact working with models of real neurons. Instead, they may more properly be called “connectionist” models, which are comprised of multiple components acting in parallel, but the components themselves are often “artificial neurones” with properties quite unlike real neurons. There are certainly exceptions, most notably the work of Eliasmith & Anderson (2003), but most “neural network” models are primarily neurally-*inspired* models, not ones that could make use of observations of real neurons to make overall temporal predictions. As the the modelling of operations down to the neural level improves, it will become more appropriate to bring in neural organization constraints. But, it may be that the basic operations underlying high-level cognitive behaviour are not themselves best described in terms of neurons. Other formulations, involving large groups of neurons, or other complex structures may be needed. For this reason, the approach taken in this thesis will not *require* that cognitive models be described in neural terms. Of course, if one wishes to do so, the methodology described herein will still apply.

As Dretske argues, we need to build these models (or be able to build them) before we can say we know how the the cognitive behaviour being described actually works. But we don't have to build them out of neurons. If we are able to build systems functionally equivalent to aspects of cognition using components that we do understand (i.e. any implemented computational model), then we can say that we understand that system.

However, until we can build those components out of neurons, we cannot say we understand *those components*. In the long run, we will want to learn to build the components out of neurons (and, indeed, to build those neurons out of atoms, and those atoms out of quarks, and so on), but in the short run we can make progress without this mechanistic reduction.

The overall approach, then, is to build models which are accurate on a wide variety of measures and situations, exhibit a wide variety of the behaviours of the real system, and are themselves composed of temporally defined operations which process information in a constrained temporal manner. These operations can then themselves be broken down for further explanation. However, one might worry that if models are not constrained to operations that are neurally implemented, it may be possible to arrive at a situation where the operations may *not* be suitable for further decomposition. That is, even though all of the other criteria are satisfied (the algorithms give correct time predictions, the system is accurate over a wide variety of situations and for different sorts of measures, and perhaps even is associated with the correct general brain regions and connectivity), there may be no way of building those operations out of neurons in such a way as to produce an accurate model.

If there are two (or more) competing models which produce identical outputs over the same range of situations, but one of them is neurally decomposable and one is not, then a decision may be made to stick with the decomposable one. However, if the non-decomposable one is a better model on these other criteria, then it is unclear what should be done. Indeed, even in a situation where the non-decomposable one is multiple orders

of magnitude less complex than the decomposable one, but equally accurate on all other measures, the former may be preferred as a model of cognition, since it may be more understandable or amenable to future development. Explaining these models themselves can then be a task for the future.

3.3.3 What Should Be Equivalent

Throughout this description of how to establish the equivalence of models and reality, the question of where to stop this process has not been addressed. After all, it is clear that there will always be certain differences between them. The models are implemented in computer code, while the real system is an interacting combination of neurons, muscles, and sense organs. The models produce numerical values as outputs, while the real system pushes buttons, pulls levers, or verbally answers questions. The models are completely controllable, while the real system has numerous unknowable individual variations due to past experiences and genetics.

This raises the question of what *aspects* of a model need to be equivalent to the real system for us to consider them, in some sense, the same. To a certain degree, this has already been addressed in Pylyshyn's notion of strong and weak equivalence. His strong equivalence occurs when the same internal algorithm is used in both systems, while weak equivalence only requires that the overt behaviour is the same (i.e. that the outputs are the same for the same inputs). This is the basis of the algorithmic equivalence discussed in section 3.1.2, above. In order for the same algorithm to exist in both the real system and the computational model, the same components for the algorithm must be available, but

they can certainly instantiate the components of that architecture in completely different ways. The idea, then, is that a model needs to capture the process of events occurring internally to the system being modelled. This sequence of events can be seen as an algorithm describing exactly what is happening at each step, built out of smaller underlying components. Such a model is strongly equivalent to the real system, and is the ultimate goal of modelling research.

An important observation in such a situation is that the differences between model and reality in terms of how the underlying components are implemented is not important. A computer model may use transistors and silicon, while the human brain may use neurons and neurotransmitters, but we can still say they are following the same algorithm. This is also true even at a higher level: if an algorithm has a step that requires two numbers to be added together, it is irrelevant whether that addition is performed by the n-bit addition algorithm embedded in modern microprocessors, or by some combination of voltages and neural firing rates, as may be seen in living brains (Eliasmith & Anderson, 2003). If both implementations are being used in the same ways by the algorithms, with the same results, then these differences can be seen as inconsequential. All that is required is that there must be some way for the brain to do what each of the components in the algorithm do, and to do so within the time frame specified by the model, as discussed in the previous section.

In other words, when modelling a particular cognitive phenomenon, the choice of underlying operations places limits on the degree of equivalence of the model. These particular operations are not themselves meant to be occurring in a way which is

equivalent to the real system; only the overall algorithm (and behaviour) is meant to be the same. Certainly, one could create models for these underlying operations, detailing how they might be broken down into even lower-level operations and lower-level algorithms, down to, eventually, neurons (or molecules or atoms or quarks). However, such a breakdown is not required. By setting these limits, the model is explicitly stating that such lower level implementational details *do not matter*. Different implementations do not affect the functional or computational aspects of the overall system, and so can safely be ignored.

While the above idea occurs frequently in cognitive science (most famously as the tri-level hypothesis; Marr, 1982), there is also a second limitation on equivalence implied by most (if not all) cognitive models. Since the system being modelled is so complex, cognitive models do not just differ from the real system in terms of how the algorithmic steps are implemented; they also differ in terms of *what other systems they are connected to*. In the RELACS model (described previously in section 2.3), a decision is made to press one of two buttons. However, in order to express this as an overt behaviour, this system must be connected to a motor control system that is capable of moving a hand to the physical location of the button and pushing it. Such a system requires massive integration of sensory feedback from both hand and eye, and is clearly not a part of the RELACS model. Similar issues arise for the perception of the buttons themselves, the interpretation of the numerical rewards, and even for the understanding of what task one is being asked to perform. None of these are part of the model, so clearly this model does not present an algorithmic account of how these parts of the cognitive system work.

Instead, the model is an algorithmic account of one particular component (or perhaps a few components) of the system.

However, the model is not just an account of *any* component in the system. What is special about the component(s) for which the model does provide an algorithmic account is that they are the ones 'responsible' for the overall behaviour of interest. That is, a model can safely ignore many components of the overall cognitive system. The overt behaviour of the RELACS model is not affected by the how the motor planning system works, or by how the perceptual attention system works. In other words, a good model not only can ignore details of how the underlying operations work, but can also ignore details of how the other components with which it interacts work.

There is, of course, a limit to this freedom to ignore details at lower levels and across different components. Given the complex and dynamic nature of cognitive systems, it is always possible (and, indeed, likely) that details of how these ignored components work will, in fact, affect the overall performance of the system. It is possible that details in how the perceptual system identifies the buttons to be pressed, or reads the reward feedback, will affect the decision making process. This is the case even if RELACS does happen to be a perfectly correct model. Indeed, the same is true of the implementations of the underlying operations within any model: unexpected variations in how addition is accomplished by neurons could certainly affect the overall behaviour. Thus, by not modelling these aspects of the system, a variety of sources of mismatch (or 'error') are introduced *no matter how good the model is*.

Since this state of affairs is the case for *all* modelling work, there needs to be some way of accounting for it within the modelling process. In particular, this can be dealt with by *quantifying* the difference between the model and reality. This gives a maximum limit of how much effect these aspects of the system that are ignored by the model can have on the overall behaviour. That is, if the model found to be accurate to a certain degree, then the best case scenario is that the model is, in fact, strongly equivalent to the real system, and the source of the discrepancies between model and reality are due to these ignored external aspects of the system. This is the situation, for example, in mathematical modelling in physics, where a model that ignores friction may be accurate to a large degree. These aspects such as the other components in the system, or the details of the implementations of the underlying operations, cause only a limited (and numerically specified) amount of error. If a more accurate model is needed, then these other components must be added into the model.

That said, it must be remembered that *there is no way to distinguish the three possible sources of error*. Any differences in behaviour between the model and reality could be due to a) differences in how the operations are implemented; b) differences in how the other components interact with the component(s) being modelled; or c) an incorrect algorithm. In other words, one cannot know for certain whether the model is, in fact, strongly equivalent to the real system (i.e. follows the exact same algorithm). It is always possible that the model has the wrong algorithm, rather than the differences being due to other aspects. This is not, of course, a new problem for science: Newton's Law of Gravitation seemed to be very accurate for a long time (never, of course, perfectly

accurate), until it was superseded by a fundamentally different theory via Einstein's Special Relativity. During that time, there was no way to know whether the differences between prediction and observation were due to a) different ways of determining mass or momentum, b) interaction effects with other forces, such as electromagnetism, or c) a more fundamental problem with the idea of a force based on the mass and distance between objects. This ambiguity is always the case for science, but at least the size of these discrepancies can be determined and reduced over time.

The best scientific procedure, then, is to pay close attention to the accuracy of the models. All models will have some degree of error between their behaviour and that of the real system. By quantifying this error, an upper bound can be placed on the size of this error for the particular situations and tasks being examined. Since this error could be due to the operations, the other components, or the algorithm itself, it is important to explicitly identify these assumptions that are being made by the model. It must be made clear that a model such as RELACS does not provide an account of the perceptual or motor aspects of its behaviour. Indeed, it does not provide an account for how the previous experiences are stored in short (or long)-term memory that allow it to adapt based on the feedback. These aspects of the model are simply assumed by the creators of the model to be simple (i.e. can be modelled by any simple mechanism) and thus to *not affect* the overall behaviour to a significant degree. The amount of error observed indicates how much of an impact such assumptions may be making.

A model can thus be seen as a way of specifying that some particular algorithms are responsible for most of a particular observed behaviour. Differences will always be

apparent between model and reality. The goal is to gradually reduce the amount of error. To do this, the various assumptions built into the model must be identified, and then studied in future models.

To return to the original question starting this section, this also provides an explicit criterion for what aspects of a model must be algorithmically and behaviourally equivalent to the real-world situation. What is required is for the model to provide an algorithmic account of *whatever components are necessary for the model to be sufficiently accurate*. The very fact that ignoring a particular aspect of the real-world situations still results in an accurate model can be taken as evidence that this aspect does not matter. Any aspect of the real world which *can* be successfully ignored by a model is therefore not an important aspect of the situation. The model can make the simplifying assumption that, for example, there is a direct mapping between the cognitive system deciding to press A and the actual physical pressing of the button. If the model is accurate, then this is evidence that there is no complex feedback that causes the physical nature of this process to impact the cognitive system.

It is certainly possible, of course, that as models are applied to more domains, this interaction between the physical process and the cognitive algorithm will become significant. For example, in a situation where one button requires significantly more effort to press, or is farther away, these aspects will likely become important to the overall decision making. A model which ignores these aspects will not be behaviourally or algorithmically equivalent. Resolving this by adding new aspects to the model will then be part of the process of expanding it and making it more comprehensive. When

this is done, there will still be aspects of the model which are not equivalent to the real world situation. The model may have a numerical value indicating the resistance of the button, while in the real world the resistance may be governed by the electrostatic forces within the spring behind the button. If the model is successful, however, then this can be seen as a claim that there is a simple mapping between this numerical value and the electrostatic forces, and the details of this mapping are irrelevant. If this does turn out to be the case, and no situations are found where the details of this mapping impact the real-world behaviour, then the model does not need to be expanded to include such details. This stops models from having to include *everything*.

3.4 Summary

3.4.1 State of the Art

- Models should behave equivalently to the real system across a variety of types of situations. There should also be different types of behaviour examined, including such things as how long actions take, what sorts of errors are made, and the effects of learning over time.
- To establish that a model is equivalent to the real system in terms of the algorithm used, observations of time can be treated in a special manner. If an algorithm requires more steps in one condition than another, then the observed reaction times in that condition should be larger than those in the other condition.
- Establishing that a model has the same structure as the real system is more problematic. Particular processes within a model may not be localized to

particular physical structures. Neural structures as identified by neuroscience may not be at the right grain size to map onto the structures in the model.

Structural equivalence is desirable, but not necessary for cognitive research.

Since a model is built from components, those components must be understood before the whole system can be said to be completely understood. However,

those components do not have to be accurate models of the real system's

components. Understanding how the components of a model function can happen after the overall system is successfully modelled.

3.4.2 Background

- There are three mutually exclusive ways of interpreting models that have been shown to be equivalent. If one believes science is just about making predictions, establishing equivalence across a wide range of situations leads to models more able to make predictions. If one believes scientific models are the basis for analogies, establishing equivalence across situations strengthens this analogy between model and reality. If one believes identifying concise methods for describing behaviour is the basis for calling something real, establishing equivalence determines the scope of the pattern responsible for the observed behaviour. This thesis is agnostic as to which of these interpretations should be followed. In every case, however, there is a strong desire to establish models that exhibit equivalence on a wide variety of types of measures, including basic behaviour, reaction times, and error rates.

3.4.3 Contributions

- Pylyshyn's approach to algorithmic complexity was expanded, arguing that numerical time predictions can be produced from computational models. This involves forcing models to be process models (i.e. to have a consistent internal time), and to produce reaction time responses based on explicit time requirements for the internal steps. As an example, it was shown that the RELACS model uses information at a given time that is not available to it until much later. This means that it cannot be a process model of human cognitive behaviour in the repeated binary choice task. This problem must be addressed for it or any model to be a plausible or even possible cognitive model, regardless of its behavioural equivalence.
- It was argued that models implicitly indicate what aspects of the process are important for the system being modelled. Models are always an approximation of the real system; therefore there will always be differences to some degree. Many aspects of the real system will be ignored by the model. Any aspect of the real world which can be successfully ignored by a model (with it still achieving the desired degree of accuracy) is therefore not an important aspect of the situation.
- Three separate possible sources of error in a model were identified: incorrect algorithms (i.e. a lack of strong equivalence), particular aspects of the real system not being included in the model, and implementation differences between the model and reality. A comparison to physics models was used to note that there is no method to reliably distinguish between these sources of error.

4 Developing a Methodology: Measurement

In the preceding chapter, a variety of considerations were identified for evaluating the equivalence between a model and the system that is being modelled. Of these, the behavioural and algorithmic equivalences were identified as the most important.

Behavioural equivalence is based on the numerical match between a set of measures made on the performance of the model, and a similar set of measures made on the real-world system. Algorithmic equivalence is established indirectly, by the numerical match of timing measurements, with the constraint that it must be a process model.

Importantly, this means that *all equivalence is established by numerical comparison of sets of numbers*. Both behavioural and algorithmic equivalence have been reduced to taking certain sets of numbers from the model and a comparable set of measures from the real-world subjects, and determining whether or not they match. However, in order to do this, a more complete definition of what is meant by a *match* between these sets of numbers is needed.

To develop a method for this numerical comparison, this chapter first discusses how mathematical models in physics deal with a similar problem. This then leads to an investigation of possible statistical measures that would be suitable for the cognitive modelling situation. The chapter concludes with a discussion of the problem of *modifying* a model by adjusting parameters.

4.1 Background: Modelling in Physics

One further source of information on these various modelling issues is to examine how

they have been addressed in traditional physical sciences. If these issues have been successfully addressed therein, then the practice of computational modelling in cognitive science may benefit from these same techniques. Indeed, as will become apparent, many of the issues are identical, with the exception that computational modelling allows much more complicated models to be feasible.

The main example to be considered is Newton's theory of gravitational attraction, summarized by

$$F = \frac{G \cdot m_1 \cdot m_2}{d^2}$$

This formula allows us to determine the force of attraction (F) between two objects of known mass (m_1 and m_2), separated by a particular distance (d). To make use of this theory, we use the formula to create a predictive mathematical model that is *customized to the particulars of the situation to which we are applying the theory*. Since this is a mathematical model, we do this by replacing the variables within the model with the relevant values and performing the indicated calculation.

For example, if we have a 5 kg object near the surface of the Earth, then we can let m_1 be 5 kg, m_2 be 5.9742×10^{24} kg (the mass of the Earth), d be 6.3781×10^6 m (the distance from the surface of the Earth to its centre), and G be 6.6730×10^{-11} m³/kg s² (the universal gravitational constant). The result is 48.999 kg m/s², which is the model's prediction of the force downwards on the object due to gravity.

Testing this prediction is difficult, since directly observing *force* is not possible.

However, Newton's second law of motion defines the relationship between force and acceleration (a).

$$F = m \cdot a$$

Using this formula, and assuming that the Earth's gravity is the only significant force acting on the object, the acceleration can be predicted as well. Mathematical rules allow us (in certain situations) to reorganize a mathematical model into other forms, while ensuring that the predictions remain identical. Thus, we can change the model to the following form.

$$a = \frac{F}{m}$$

We again substitute the appropriate values into the equations, resulting in a predicted acceleration of 9.8 m/s^2 , which is in accord with the observed behavior of the real object.

The intent behind framing the use of mathematical models in this manner is to highlight the parallels between the mathematical process and the computational modelling process. To continue the analogy, it is worthwhile to examine three aspects of such modelling: the use of computational models, the application of numerical comparison, and the role of parameters.

4.1.1 Computational Models

When models in physics become increasingly complex, physics models start to become similar to the computational cognitive models, since the pure mathematical proofs that

are familiar in physics become less applicable. That is, when the situations become too complicated for the mathematics to be solvable, a process strikingly akin to computational modelling must be performed. Understanding the success of this process in physics may be useful for understanding computational modelling in general.

This situation can be demonstrated by considering the idea of predicting the *acceleration* of an object due to gravity. Much like force, acceleration is not directly observable. However, the general framework in which the theory of gravitation is embedded defines acceleration as the change in velocity over time, and defines velocity as the change in position over time. Thus, given a known acceleration, we can predict the position of an object over a period of time, and given known positions, we can determine what pattern of acceleration would give the same result. That is, in order to evaluate the predictions of this model, we need accurate information about the temporal aspects of its predictions.

For simple cases, where the acceleration does not change significantly, the positions can be determined by basic mathematical manipulation of the definitions of the terms. If v_0 is the initial speed of an object, and it accelerates at a , then the resulting velocity (v) and distance travelled (d) at a particular time t are given by the following formulas.

$$v = v_0 + a \cdot t \qquad d = v_0 \cdot t + \frac{a \cdot t^2}{2}$$

However, if the acceleration due to gravity *changes* significantly over time (as it would for planets or comets orbiting the Sun), then the matter is more complicated. The issue is that the acceleration is due to the force of gravity, which is dependent on the distance

between the objects. The acceleration causes *changes to the distance*, which in turn affects the acceleration.

This feedback loop leaves matters in a strange situation. We have a mathematical model of the phenomenon, *but we may not be able to use mathematics to determine its predictions*. That is, the resulting mathematical formulas have no known exact mathematical solution. The theory of gravitation allows us to create models of the acceleration of Jupiter due to the Sun at different points in time, but how can these be combined to produce an overall path for Jupiter?

The approach for dealing with this problem is to divide the overall period of time of interest into small intervals. Within each of these time intervals, we assume that the acceleration is constant. We can assume that the force due to gravity will not change significantly within that amount of time, and so we can make use of the simple formulas to determine the position and velocity during that interval. Then, given the new distance, we can use the gravitational formula to determine the new acceleration for the next interval.

Performing the calculations in this manner would clearly require inhuman patience and care. Furthermore, the results would change depending on how large the intervals were chosen to be. Instead, the development of the theory of gravitation was accompanied by the mathematical technique known as *calculus*. The idea here involves directly determining what the answer would be *if the intervals were reduced to an infinitely small size*. This remarkable short-cut then solves the problem, allowing the predictions to be

determined without a tedious process of gradual calculation.

Suppose that at a given time the position and velocity of a planet can be determined, and that the force is known. Then, according to Newton's laws we know the change in velocity during a short time interval. Knowing the initial velocity and its change, we can find the velocity and position of the planet at the end of the time interval. By a continued repetition of this process the whole path of motion may be traced without recourse to observational data. This is, in principle, the way mechanics predicts the course of a body in motion, but the method used here is hardly practical. *In practice such a step-by-step procedure would be extremely tedious as well as inaccurate.* Fortunately, it is quite unnecessary; mathematics furnishes a short cut, and makes possible precise description of the motion in much less ink than we use for a single sentence. (Einstein & Infeld, 1938, page 31, emphasis added)

Unfortunately, the calculus technique described above *only works for certain situations.*

If we consider a model of three planets interacting due to gravity, then the mathematics are *intractable* (i.e. there is no known short-cut). Fortunately for physicists, there tends to be only one dominant gravitational force for most celestial bodies (the force of gravity between the Earth and Jupiter is much less than between the Sun and Jupiter, so the effects of Earth and the other planets can safely be left out of such a model). This allowed astronomy to proceed with a reasonable degree of accuracy without such details.

While there are a number of attempts at mathematical tricks for dealing with special cases of this *three-body problem*, the only general solution is to follow the original approach of dividing the time into small components and performing iterative calculation. While the result of this is dependent on the size of the time interval, there are methods for determining how large such a variation could be, and so the amount of uncertainty due to this approach can be known. This is fortunate, since *most complex models are mathematically intractable.* As models get more complicated, this iterative technique is

required for determining their temporal predictions.

In other words, applying physical laws to complex situations requires the creation of computational models based on the underlying mathematical model. This process allows physics to deal with more complex situations. Thus computational modelling in cognitive science can be seen as a natural progression of this tool.

4.1.2 Numerical Comparison

The next aspect of mathematical modelling which bears closer examination is the process of confirming the model's predictions. If a model predicts that an object will fall 4.9 meters in one second, and measurement reveals that it only fell 3 meters, then it may be reasonable to conclude that the model was not appropriate for the situation. If, however, the object is observed to fall 4.899999 meters, then it seems our model is supported. Between these extremes, a principled method is needed for determining if the model matches or fails to match the observed situation.

In some situations (especially when the iterative approach is used for determining the predictions over time), there is a method for mathematically calculating the maximum 'error' possible due to choices such as the size of the time interval. This can give a lower bound on the allowable difference between the prediction and the final result (i.e. if the observed error is greater than the predicted maximum error, then the theory can be considered inaccurate). Similar considerations can be made due to the variability in the initial measurements used to set the parameters of the model (such as the object's mass).

As an example of this process, the Nobel Prize in Physics in 1965 was awarded to Sin-

Itiro Tomonaga, Julian Schwinger, and Richard Feynman for their work on Quantum Electrodynamics. The key measurement which established the quality of this theory was the magnetic strength of an electron. The real-world value for this is 1.0011596523 ± 3 , while the prediction from the theory is 1.0011596524 ± 2 (Feynman, 1979). Interestingly, both of these values have confidence intervals, due to the complexity of the experimental measurement and the complexity of the mathematical calculation.

The result is a comparison measurement between model and reality based on how different the model and reality could be. In the above case, the maximum possible error between the model and reality is 0.0000000006. If this approach is to be applied to computational cognitive modelling, then a similar sort of measurement may be useful.

4.1.3 Parameter Values

Finally, more must be said on the determination of the particular values being used for substitution in the models. How do we know that the mass of the Earth is 5.9742×10^{24} kg, or that its radius is 6.3781×10^6 m? Indeed, what does it mean to say that the object has a mass of 5 kg? Using only geometry and the position of the sun at the same time in two different cities, the Earth's radius can be determined to a reasonable degree of accuracy.¹² The mass of reasonably sized objects is determined by defining one particular reference object to be 1 kg, and then using a balance to determine that the mass of a given object is, in this case, five times the mass of the reference object, or 5 kg. This

¹² This method is usually credited to the ancient Greek scholar Eratosthenes, but it is likely that it was known to others previously.

method, however, is not applicable to determining the mass of the Earth.

Indeed, for 120 years after the development of the theory of gravitation, the mass of the Earth was unknown. Importantly, *the theory was useful without knowing absolute values of all the parameters*. Due to the simplicity of the relationship between G , m_1 , and m_2 , they can be combined into a single parameter, and for any given situation, we can determine what value for this combined parameter *best fits*. For example, if we use the formula to predict the influence of the Sun on Jupiter, we do not need to know the mass of either. Instead, we can observe the path of Jupiter for a short period of time, determine the force applied by the Sun that would be required to result in such a path¹³, and then determine for what value of this combined parameter the model would give the same result. Once this value is determined, it can then be used in all future predictions of the gravitational influence of the Sun and Jupiter. In other words, *the model parameters are fit to one situation, and then applied to other situations*.

However, if the mass of the Earth was known, then we would be able to apply the formula without this initial fitting stage. This can, of course, be seen as doing parameter-fitting on the individual parameters in the formula, instead of the combined context-dependent parameter. Knowing these values would allow for a more rigorous test of the theory, and would lead to more general uses. In particular, it would give us a value for G , the gravitational constant. This is a parameter which has *the same value in every application*. In fact, in terms of using the theory to produce models, we can consider that the theory itself *includes* the specification of this parameter. Taking this approach, James

¹³ Using geometry and Newton's laws of motion.

Cavendish evaluated G by devising a situation where the gravitational attraction between two reasonably sized objects could be measured¹⁴, and used this result to determine the best-fitting value of G . This, in turn, could be used to determine a suitable value for the mass of the Earth. Importantly, we must note that these evaluations are done *assuming the theory is correct*. They give us values that can be used in the creation of other models in other situations. These other models can then be considered predictions of the underlying theory, which includes the specifications of these constants.

A more detailed theory, however, may indicate particular values for certain parameters. Once a value for G was included within the theory of gravitation, it could be used in new situations *without the stage of first customizing the model*. This is clearly a desirable property for a cognitive theory as well. The theory could specify a particular value for a parameter, or even indicate a range of values, meaning that the model will be suitable *no matter where in that range the parameter is set*.¹⁵

4.2 State of the Art: Numerical Comparison in Modelling

In all of the preceding discussion there has been the assumption that there is some way of determining how “close” the match is between the measurements on the model and those those made from observations in the real world. This is also true in the physics example given in the previous section. Exactly how this is to be done is far from a trivial problem, and a wide variety of mathematical techniques have been proposed for this process.

14 Using a large barbell suspended by a string placed near other weights.

15 Merely saying that there will be a parameter value somewhere in some range for which the model will give accurate predictions puts us back to the original situation of needing to customize the model before applying it.

Axelrod (1997) discusses three sorts of comparisons.¹⁶ *Numerical identity* is a situation where the two things being compared are completely indistinguishable in terms of their exact behaviour. This would be a model which exactly matches the observed behaviour in every respect, right down to the exact errors that were made and which order they were made in. All of the raw observational data would thus be exactly the same for both the model and reality. *Distributional equivalence* is a comparison based on the analyzed statistics. Here, consideration is given to statistical matching; although the details of the performance of the model may differ from reality, the overall *rate* of error may be the same. Importantly, this measure does not just look at the statistical mean performance. Instead, the shape of the distribution curve should be the same. This would then include such measures as the standard deviation, mode, median, skew, kurtosis, and so on. *Relational equivalence* is a still weaker measure, which involves the qualitative behaviour of the models. Here, the question is whether the behaviour follows the same pattern, in the sense of one measure always being higher than another measure, or always decreasing in response to some particular change in the situation.

For the purposes of cognitive modelling, numerical identity is too stringent a criterion. Axelrod proposes it in the context of comparing two different implementations of a computational model, and in that context requiring this sort of exact replication can be a useful debugging tool. However, for the foreseeable future cognitive science will be quite satisfied with distributional equivalence, and that will be the criterion used for the remainder this thesis. For an exception to this, see chapter 13 on Qualitative Models,

¹⁶ Axelrod considers model-model comparisons, as opposed to model-reality comparisons, but the point is still the same.

which will rely on relational equivalence.

The goal, then, is to take a set of numerical measurements from experimental results and a set of numerical measurements from a computational model and statistically compare the two sets of numbers. There can be only one such measurement (for example, the subject's reaction time in an object discrimination task, or the number of errors made in an arithmetic task), or there can be a collection of measurements (from different experimental situations, or measuring different aspects of the behaviour). The raw data from the real-world measurement will be a series of numbers. The majority of cognitive models have some sort of stochastic aspect to their behaviour, since this is necessary if the model is to capture the variability in the human data. Thus, running the model multiple times will also produce a series of numbers. What is required is a technique to take these series of numbers and determine whether they are distributionally equivalent to some desired degree.

4.2.1 Correlation

The simplest numerical measure is to find the correlation between the two sets of numbers. This is an indication as to whether the model's behaviour varies across different measurements in a similar manner as the observed behaviour does. It ranges from -1 to 1, with zero indicating that there is no relationship between the sets of numbers, a 1 indicating that whenever one value changes, the other also changes by a linearly related amount. A -1 indicates that the change is in the opposite direction. Given a collection of n real-world measurements (x_i) and model measurements (y_i), the

correlation (r) is calculated as follows:¹⁷

$$r = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}}$$

For example, Connell and Keane (2006) investigate a model of plausibility judgements by measuring the correlation between the model's output for 60 different situations and the mean human judgement for the same situations. That is, a variety of scenarios such as “The dress snagged on a nail. The silk ripped” were presented to human participants, who were then asked to give a numerical rating from 0 to 10 on how “plausible” these situations were. These same scenarios were also presented to a computational model, which outputted numerical ratings based on background knowledge and characteristics of the paths of inference required to process the scenario. The mean of the human data was taken and plotted against the output of the model, as shown in Figure 4.1, below. Their model has no random behaviour, and always outputs the same value for each scenario. The drawn line is the linear regression of the data, and the indicated correlation (r) is 0.77.¹⁸ They indicate that the model's “estimates correlate strongly with the participants' mean plausibility judgements per scenario, and regression analysis suggests that the model could be used as a successful predictor of human plausibility ratings” (Connell & Keane, 2006, p. 109).

¹⁷ As an important implementation note, directly implementing this formula in a computer program is known to be highly unstable, and most naïve approaches produce incorrect results due to rounding errors. It is recommended that researchers use existing implementations of the correlation algorithm, rather than generating their own.

¹⁸ These values were hand-copied from the figure in the original paper, so there are likely to be some differences from the actual data. In the original, the r^2 value was 0.60 and the r value was 0.776.

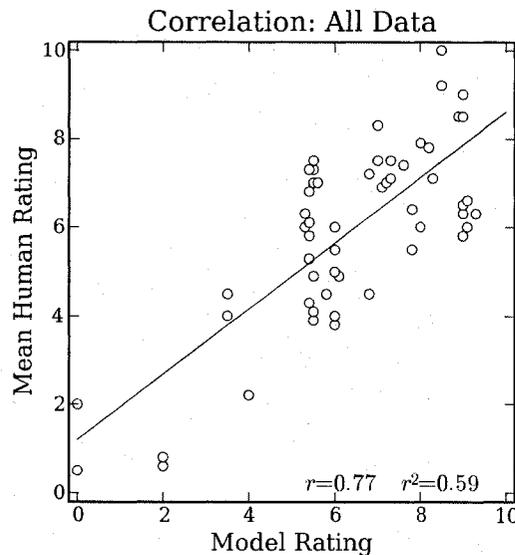


Figure 4.1: Scatter plot for comparison of model and human data. Each point is a different scenario. Data from Connell & Keane, (2006, Figure 7).

The description of this sort of correlation as “strong” corresponds with a suggestion by Cohen (1988) that, in the psychological sciences, a value above 0.5 is strong, between 0.3 and 0.5 is moderate, and between 0.1 and 0.3 is small. However, it is unclear how this value should be interpreted in a modelling situation. One way to approach this is to note that the r^2 value, under the assumption of normally distributed data, indicates what proportion of the variation is explained (or, more correctly, is predictable based on a linear fit) by the model. That is, in Figure 4.1, 59% of the variance in the means of the human ratings (i.e. the differences between how participants, on average, rated the difference scenarios) is predictable given the model.¹⁹ One can also use this measure to judge between two competing models, as the one with a higher correlation will be

¹⁹ Since no information about the individual variability is used, and since the model does not exhibit any variability itself, it is unable to account for any of the between-subjects variation.

preferred. Connell and Keane use this approach to find the best parameter settings for their model; adjusting values to find the highest correlation.

However, by grouping all of these measurements together, the correlation measure is only informative about the *large-scale* features of the data. In this situation, important small-scale differences are being masked by the overall correlation. In the actual experiment, four different types of scenarios were presented to the subjects (identified here as types A, B, C, and D). Figure 4.2 shows what happens if each of these situations are analyzed separately.

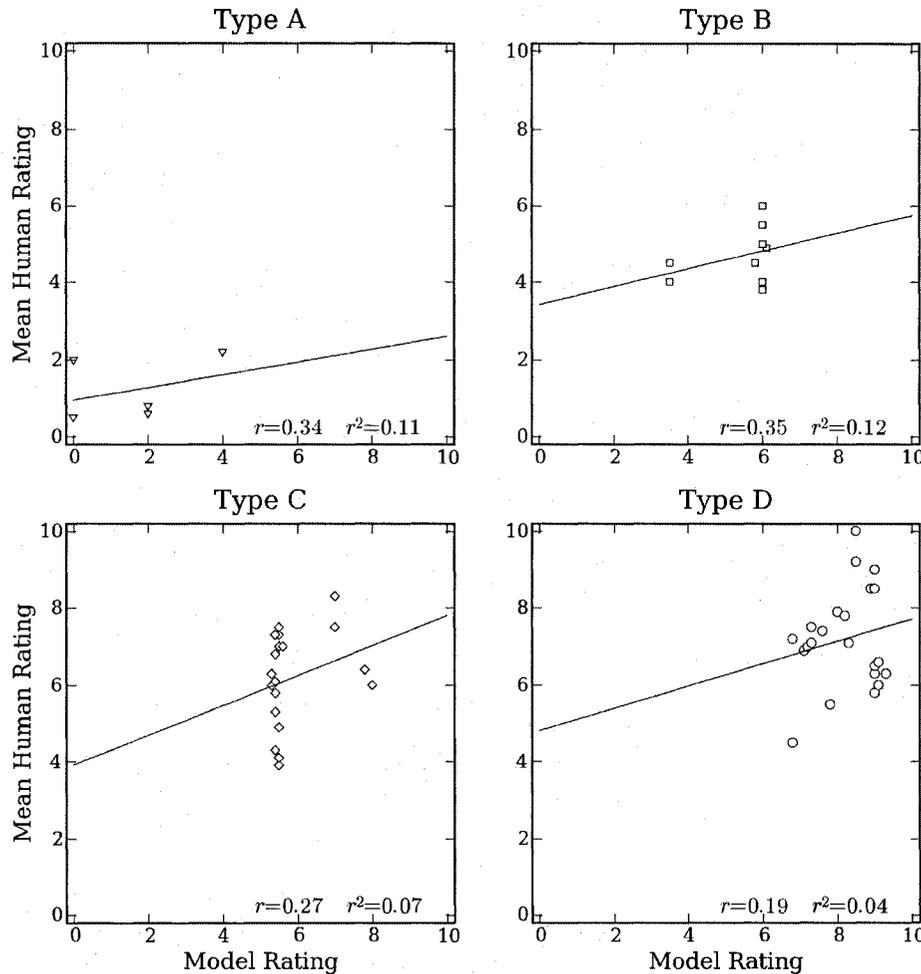


Figure 4.2: Scatter plots for comparison of model and human data. Each point is a different scenario, separated into the four categories of scenario investigated. Data from (Connell & Keane, 2006, Figure 7).

Once the scenarios are separated in this way, the explanatory power of the model seems lessened. Within each category, there is only a weak to moderate correlation between the model's behaviour and that of the human participants. While it is true that the model “rated certain [Type C] scenarios as more plausible than some [Type B] scenarios, and certain [Type B] scenarios as more plausible than some [Type C] scenarios” (Connell & Keane, 2006, p. 111), as did the participants, *it did not do so for the same ones*. This

calls into question the conclusion that “[s]uch variability in [the model’s] ratings parallels the variability seen in human experiments and demonstrates the model’s success in capturing the range of group behavior” (ibid). In other words, the model might be better considered one which is able to distinguish between these four situations for plausibility judgements, but does not provide detailed information about the processes involved in distinguishing *within* such classes.

4.2.2 Root Mean Squared Difference

An alternate, and probably the most common, approach for comparing model data to experimental data involves calculating the mean squared difference. The idea here is to represent the size of the average difference between the model and reality. This value gives a directly interpretable value; instead of worrying about whether a correlation of 0.6 is “weak” or “strong”, an average difference of, for example, 68 milliseconds would indicate that the model’s predictions are usually accurate to around 68 milliseconds.

However, there are a variety of possible measures that could be made which fit this description. The most common of them is the Root Mean Squared Difference (RMSD). This measures how different the two data sets are by taking each of the differences between the model and subject data, squaring them, averaging them together, and then taking the square root. The Mean Squared Difference (MSD) is also common, and is identical except for skipping the final square root. Simply taking the Mean Difference (MD) but just averaging the differences is rarely, if ever, used. Also rare, but easily interpretable, is the Mean Absolute Difference, which is simply the average of the

absolute differences between the values. The formulas for all four are as follows, with x_i and y_i being the various measures for the model and reality, and n being the number of different measures considered.

$$RMSD = \sqrt{\frac{\sum (x_i - y_i)^2}{n}} \quad MSD = \frac{\sum (x_i - y_i)^2}{n} \quad MD = \frac{\sum (x_i - y_i)}{n} \quad MAD = \frac{\sum |x_i - y_i|}{n}$$

For example, the data presented in Figures 4.1 and 4.2, above, result in the measurements shown in Table 4.1. The RMSD measure is the most commonly used, as it has a variety of mathematical features which make it suitable for proofs, which will not become relevant here. The MSD is for most purposes equivalent, although it has the disadvantage of not being in interpretable units, so it cannot be directly compared to the experimental situation. Also, less scrupulous individuals may take advantage of this fact to give the appearance of a closer match; a RMSD value of 0.15 seconds would become a MSD value of 0.0225, which may seem psychologically closer. The MD is problematic, as many large over-estimations and many large under-estimations would cancel each other out. The MAD is simple and does not give more emphasis to points that differ by more than others (as RMSD and MSD do), but is not generally used.

	Root Mean Squared Difference	Mean Squared Difference	Mean Difference	Mean Absolute Difference
All Scenarios	1.790	3.204	0.589	1.404
Type A	2.085	4.349	1.227	1.482
Type B	1.604	2.573	-0.247	1.405
Type C	1.501	1.501	0.878	1.211
Type D	1.506	1.506	0.440	1.400

Table 4.1: Various difference measurements for comparing model and real data.

Each of the values in Table 4.1 can be seen as an alternative to the r and r^2 values discussed in the section on correlation. Both the RMSD and the MAD are directly interpretable, giving the indication that the model generally produces plausibility judgements that are within 1.4 – 1.8 of the values given by the human participants.

Furthermore, this is roughly true of each of the individual categories of scenarios as well, although the Type A group is off by slightly more. This sort of measurement provides a way of indicating the size of the difference in behaviour one expects to find when using this model.

Unfortunately, it is also true that the *averaging* process causes a significant loss of information. Looking at the graphs in Figures 4.1 and 4.2, it is not uncommon to see values that are much farther away from the model prediction than the average. Indicating that the mean difference is of a certain size does not indicate the model's worst-case performance. Indeed, it is possible to arbitrarily improve the RMSD value simply by *adding more of the sorts of measurements the model is good at*. In this case, adding a large number of measurements in the Type C category could make the measurement improve considerably, since these are the situations that the model does best on. This is a

danger from any approach which averages over measures.

In the RELACS model (Erev & Barron, 2005), MSD is used as the main measurement. The best fit model had a MSD of 0.0036 over the 40 different experimental conditions considered in the core of their paper. However, this result masks the fact that the model performed significantly better in some situations than in others. For example, the model consistently exhibits identical performance for positive and negative rewards, while the human participant data points gathered indicate a variety of differences. These are not discussed in their paper, as they only consider the average performance over all the measures.

4.2.3 The χ^2 Test

Both the Correlation and the Root Mean Squared Difference measures mentioned above share a fundamental problem. A high-quality model should produce data statistically indistinguishable from the real system being modelled. However, both of the measures examined so far only involve the *mean* of the behaviour. That is, if the human participants exhibited an average reaction time of 500 milliseconds, but varied by plus or minus 50 milliseconds, we should expect the model to do the same. However, both Correlation and RMSD would give the same results if the model varied by plus or minus 5 milliseconds, or even plus or minus 500 milliseconds. A good measure should be sensitive to situations where the model happens to vary not at all (as is the case in the plausibility judgement model discussed above), or varies by much more than is expected.

One method for establishing the correspondence between the statistical distributions of

two sets of data is the χ^2 (chi-squared) goodness-of-fit test. Here, the different possible outcomes are grouped into *bins*, and the model is used to predict how many of each group of outcomes is expected. The real-world data can then be compared to this prediction. The difference between what occurs and what was expected is determined, and the χ^2 statistic identifies whether the observed difference is greater than what would be expected by chance.

For example, consider a situation with only one measure being made (for sake of simplicity). This can be a numerical measure such as reaction time, a non-numeric measure such as which of five possible outcomes occurred, or even a simple success or failure measure. In this example, the measure is a numerical value between 0 and 10 reported by the participants. These measures are then grouped into bins, which can be of arbitrary size. Note that there is no “best” size for these bins (as the optimal size will depend on the distribution itself), and so the choice of the size of the bins can affect the results. In this case, five bins are defined, and the number of responses by the different human subjects are shown in the “Human Frequency” row in Table 4.2. In this case, most of the 20 subjects tested on this scenario reported a value between 6 and 8. This can be seen as a simple histogram. The model is also run a large number of times, and frequencies gathered for it as well, as shown in the “Model Frequency” row.

Response Bin	0-2	2-4	4-6	6-8	8-10
Human Frequency	1	2	3	8	6
Model Frequency	42	113	140	418	287
Expectation	0.84	2.26	2.80	8.36	5.74

Table 4.2: Example frequency data for different possible responses to the same experimental situation by 20 separate participants and 1000 model runs.

Given this data, the model frequencies can be used to generate the *expected human frequencies*. That is, if this model is accurate, what values should be given for the human frequency, if there were an infinite number of participants. This is calculated by simply taking the model frequency, dividing by the total number of model runs, and multiplying by the number of participants. This result is given in the “Expectation” row.

There is clearly a difference between the expectation from the model and the observed human data. However, it is possible that this difference is simply due to chance. In fact, it is highly *unlikely* that a sample of 20 individuals would result in an *exact* match to the model, as there is always some form of sampling variability. To determine whether the difference is large enough to cast doubt on the model, the χ^2 statistic is calculated. O_i is the observed frequency, and E_i is the expected frequency.

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

In this case, the resulting value is 0.102. To interpret this, the value is compared to a critical threshold value retrieved from a χ^2 table. Determining this threshold value requires finding the *degrees of freedom*, which is equal to one less than the number of

bins in the above table. There is also a correction factor that can be applied if model fitting has occurred (as fitting the model by adjusting parameters opens up the possibility of fitting to spurious aspects of the data). The effect of this is approximated by reducing the degrees of freedom by the number of parameters used in fitting. Once this is found, if the calculated value is greater than the threshold value, then this is evidence that it is highly unlikely ($p < 0.05$, $p < 0.01$, or $p < 0.001$, depending on the table) that the observed data could arise from the distribution predicted by the model.

In this example so far, only a single measurement has been considered. However, in a series of papers (Anderson et al, 2004; Anderson et al, 2005) this measure has been applied to a modelling situation where the bins are predictions of frequency of activity at different locations in the brain at different periods of time. The model predicts how much activity is in each of these bins, and the fMRI blood oxygenation levels are the observed data for comparison. In this situation, each of the bins is seen as a separate measure from the model. This approach can be used to combine multiple measures, merely by adding separate bins. However, these must be comparable measures (i.e. measures of the same sort of things, in similar units, and with similar magnitudes). An alternate approach to combining measures is discussed in section 4.2.7.

Unfortunately, there are a number of limitations to this approach. The primary concern is the reversal of statistical reasoning. If the χ^2 statistic is greater than the threshold, then the model and observed data can be considered to be statistically different. This can be seen as disproving the model. However, if the value is lower than the threshold, the conclusion is that *there is no evidence that the model should be rejected*. This is not the

same as having evidence *for* the model; it is merely a statement that the available information does not reject the model at a 95% probability level. Absence of evidence for a difference between model and reality is not evidence of the absence of a difference. Finding evidence that is consistent with a model should only be seen as supportive of that model if it is also shown that other competing models are not consistent.

With this in mind, a successful χ^2 measure is certainly a good thing for a model, this measure does not provide strong evidence for a model on its own. This is especially true when it is observed that having a smaller number of observed data points (i.e. fewer subjects) increases the likelihood that a model's χ^2 value will be less than the threshold (i.e. that it will not be rejected). However, one subtle advantage of the χ^2 test is that it explicitly avoids *over-fitting to the data*. That is, there is a rule that indicates that if one model has a χ^2 value that is less than another model, but they are both less than the threshold, then this is *not* evidence that the one model is better than the other. Since they are both under the threshold, any differences at this scale are differences that are much smaller than the sampling variability. If another experimental study was performed and the model was compared to the new situation, the χ^2 values are likely to change by up to this amount. This accuracy limitation is not something that is captured by correlation or RMSE, and will become more important for the measures examined next.

Before going on to the next approach to measurement, it is worth pausing to note that the discussion has moved from simply measuring the difference between model and reality (as per the correlation and RMSD measures) to performing a more complex analysis that takes into account the sampling process. Since the real world measurements are always

found by sampling a few representative members of a population (i.e. the particular participants involved in the study), rather than the entire set of people that could be studied, it is not correct to simply assume that the exact numerical result from the study is a perfect representation. The fact that a particular set of participants have an average reaction time of 357.642 milliseconds in some particular situation does not mean that a perfect model of the population as a whole should have an average reaction time of exactly 357.642 milliseconds. Instead, one should expect a certain, predictable amount of variation around this, and different mathematical techniques are used to determine how close is close enough, and how far away is sufficient to conclude that the model is not suitable. The χ^2 test described in this section takes this into account, as will the methods shown in the next two sections.

4.2.4 The t-Test and Analysis of Variance

The next comparison method for consideration also addresses characteristics of the distribution of the measurements for the model and reality (as opposed to considering only the mean value, as in correlation and RMSD). As with the χ^2 test, the attempt is to determine whether the model and reality can be said to statistically significantly differ (i.e. whether the observed differences between them could merely be due to random sampling error). However, instead of representing the distribution with a collection of frequency bins (i.e. a histogram), an assumption is made that the data points are *normally distributed* (i.e. shaped like a normal or Gaussian distribution). If this is the case, then the actual underlying distribution of the data can be captured by exactly two variables: the mean and the standard deviation.

This test is known as the *t*-test, which is a special case of the ANOVA (Analysis of Variance) technique. The goal of the *t*-test is to determine whether the mean of the real data is different from the mean of the model data. This is done by first assuming that the model data and the real data have the *same* mean, and then working out what the probability is of observing the what was in fact observed from both the model and reality, given this assumption. In order to do this, the *t*-test makes use of the observed variance (standard deviation) of the data to measure the degree of variability. For example, if the mean of the subject data is 5 and the mean of the model data is 6, it is unclear whether this difference is due to the model being wrong, or if it is just due to sampling error. If the variance of the actual performance is high, then perhaps the *actual* mean is 6, but when the 20 subjects were chosen, they just happened to overall have a low average. If the number of subjects is high enough, this would not be a problem, but in cognitive science research there is seldom the luxury of extremely large sample populations.

To perform the *t*-test, the following formula is used to attain an unbiased estimate of the sample variance (s^2) for the model data, and is used again for the subject data. x_i represents the individual data points for each subject (or run of the model), and \bar{x} is the overall average.

$$s^2 = \frac{1}{n-1} \sum (x_i - \bar{x})^2$$

Next, it is necessary to calculate the variance for the *difference* between the means of the two groups. Since the key criterion under consideration here is whether the population means (the means that would, theoretically, be attained if there was an extremely large

sample size) are different, and it is known that the *sample* means are, in fact, different, this decision must be made based on knowing how much variation is expected in this difference. There are two methods for calculating this, depending on whether the assumption is made that the model and reality have the same *variance* as well as the same mean. The formulas are as follows, with the subscripts 1 and 2 indicating the model and real data, and N indicating the number of samples (i.e. N_1 is the number of times the model was run, and N_2 is the number of subjects measured).

$$\text{No Assumptions: } s_d^2 = \frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}$$

$$\text{Assuming Equal Variance: } s_d^2 = \frac{(N_1 - 1)s_1^2 + (N_2 - 1)s_2^2}{N_1 + N_2 - 2} \left(\frac{1}{N_1} + \frac{1}{N_2} \right)$$

Finally, a t -value is calculated that will indicate the likelihood of encountering the observed difference between the means, if it were in fact the case that the means were the same.

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{s_d^2}}$$

Much like the χ^2 value in the χ^2 test, a table of values is used to determine whether the t -value is too large to be likely to occur by chance. Again, this requires choosing a probability threshold (usually $p < 0.05$) and determining the degrees of freedom, which is $N_1 + N_2 - 2$.²⁰ The t -test table will then give a particular value, and if the calculated value is

²⁰ This formula for the degrees of freedom assumes that the variances of the model and reality are equal as well. To avoid making this assumption, Welch's approximation can be used, although this results in a non-integer value for the degrees of freedom, making it difficult to use a table to look up t -values. This

greater than this, then it is highly unlikely (less than 5% chance) that the observed difference would occur if the model and the real world were, in fact, identical.

Initially, this test seems ideal for modelling purposes. It takes into account the variability of the individual behaviour, takes care of the sampling error encountered with the small subject pools typically available, and unambiguously identifies situations where the model and reality differ. It is also a standard test used throughout psychology, so it is a familiar tool used for cognitive research. Furthermore, if multiple comparisons are desired (as for multiple measures), the more general ANOVA approach can be adapted, so as to take into account these multiple measures in the determination of probability. If 20 *t*-tests are made at a threshold of $p < 0.05$, then on average one of them will be wrong unless this repeated measure issue is taken into account, as it is in an ANOVA. These reasons combine to make this a very popular test in psychology, and a tempting one for modellers to make use of.

Unfortunately, the *t*-test has a number of limitations which make its use problematic in this situation. The first problem is one that it shares with the χ^2 test. The goal of modelling is to find models that match human performance, but the *t*-test can only *reject* models. If a *t*-value is too high ($p < 0.05$), then the model can be rejected, since the chance of observing differences this large are less than 5%. But, if the *t*-value is lower than the critical value ($p > 0.05$), this does *not* mean the model and reality have the same means. It simply means that if they did have the same means, this would not be an uncommon result.

approach is used in most computer programs.

A further problem is that the measured t -value depends strongly on the number of samples. If only a few subjects are measured (or if the model is run only a few times), then the t -value will always be small, and so the model will never be rejected. Even more worryingly, if a *large* number of subjects are measured, or if the model is run a large number of times, then the t -value will *always* eventually become too large, and the model will be rejected. The only time this will not happen is if the model is *exactly perfect to an infinite number of decimal places*. That is, since there is always going to be some difference between the actual population mean and the actual model data (if for no other reason than that there are a limited number of individuals in the population while the model can be run an arbitrarily large number of times), this means there will always be some tiny difference between the model data and the data gathered from experiments. With a large enough number of samples, this difference will be detected.

It is worth noting that these reasons also apply to many uses of the t -test in psychology. The sample size problem can be avoided by careful power analysis, but this is not always performed. These reasons have led to a number of calls for reducing the reliance on the t -test within psychology (Cohen, 1994; Wilkinson et al, 1999; Ioannidis, 2005; Beaulieu-Prévost, 2006), including some calls to ban all of this sort of testing from published journals (Schmidt, 1996). The core of this reasoning based on the fact that the null hypothesis chosen in most t -tests (that the two data sets are exactly equal) is *always* false, there is a publication bias effect that causes non-significant results to be ignored, and in any case, the goal of the research is not to prove that the data sets are not exactly equal, it is instead to try to describe *how* they differ, and if that difference is significant and useful.

This is not information that is available via a *t*-test or ANOVA.

4.2.5 The Equivalence Test

As discussed in the previous section, the standard *t*-test determines the probability of observing a difference of a certain size if the model and real data do, in fact, have equal means. If this probability is low, then the conclusion can be reached that the model and the real data differ. This is in fact the *opposite* of what is desired. A more useful test would be one that assumes the model and real data *differ*, and then determines the probability of observing a difference of the size that is, in fact, observed. If this probability is low, then the conclusion could be made that the model is unlikely to be a poor model.

A variation of the *t*-test that follows this approach is known as the *equivalence test*. It is a new technique, and is currently used primarily in medical testing in order to determine if a new, cheaper treatment is *equivalently effective* as an older treatment (Barker et al, 2002). The mathematical derivation is similar to the *t*-test, except that the null hypothesis is that the means of the two groups of data differ *by more than a certain amount* (i.e. $H_0: |\mu_1 - \mu_2| > \theta$ rather than $H_0: \mu_1 - \mu_2 = 0$). A key aspect of this definition is the introduction of a threshold value; differences greater than this amount are deemed to be significant, while differences smaller than this amount are insignificant. Importantly, this is *practical* significance, not statistical significance. Tryon (2001) recommends that the term “statistical significance” be replaced with “statistical difference” so as to avoid this confusion.

The method for calculating an equivalence test can be easily described in graphical terms, using the idea of *confidence intervals*. Instead of considering the particular mean value produced by the model (or by the experimental subjects), a confidence interval around this is constructed. This confidence interval is the range of values that would be 95% likely to be seen if the experiment was repeated (assuming that the current data values are representative). This can be reasonably interpreted as indicating the *actual* population mean is 95% likely to be within that range.

This process can, of course, be done to the standard *t*-test. In this situation, if the confidence interval includes the value zero, then the null hypothesis cannot be rejected, because zero (indicating zero difference between model and reality) is among the values that could reasonably produce the observed difference. However, by interpreting the *t*-test as a confidence interval range, the equivalence test can be naturally described (Tryon, 2001).

The first step is to construct the confidence intervals for the mean of the model data and the mean of the experimental data. The standard method for doing this assumes that the values are normally distributed, and uses the table of *t*-values used by the *t*-test (in the next section, a method will be described which does not make this assumption). The value t_{95} is the critical value from the *t*-table with $p < 0.05$, and s^2 is the sample variance, as calculated above.

$$CI_{95} = \bar{x} \pm t_{95} \sqrt{\frac{s^2}{N}}$$

Next, a modified confidence interval range is constructed which corrects for the comparison process. This slightly shrinks the confidence intervals in a way that will be useful for the final step. It should be noted that skipping this shrinking step results in a more conservative estimate, and the greatest amount of shrinking possible is $1/\sqrt{2}$, which occurs when the variances of the model and reality are equal.

$$CI_{95,corrected} = \bar{x} \pm t_{95} \sqrt{\frac{s^2}{N} \frac{\sqrt{s_1^2 + s_2^2}}{s_1 + s_2}}$$

Given two such corrected confidence intervals, it is possible to determine the *maximum difference* between them. That is, one can identify the two points farthest apart overall.

This process is demonstrated graphically in Figure 4.3.

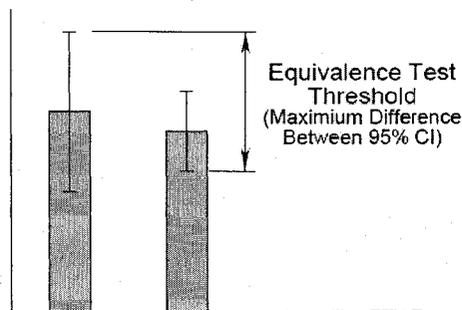


Figure 4.3: Combining confidence intervals to determine the maximum likely difference between two sets of data

This resulting value will be referred to as the *equivalence test threshold*. This indicates the greatest amount of difference there could actually be between the model and reality and still observe the amount of difference that was found, with 95% certainty. (More correctly, the conclusion that the model and reality differ by less than this amount will be incorrect at most 5% of the time.). If this total difference is less than the previously

determined threshold for significant difference, then the model and the real system are *statistically equivalent*.

This statistic results in a single, interpretable value for comparing the average model performance with the average subject performance, on a given measure. This value is a worst-case scenario indication of how *bad* the model could be. This is not an indication that the model exactly matches the subjects (as no model will ever *exactly*, to an infinite number of decimal places match the subjects). Instead, it is an indication that the model is not likely to be in error by more than this amount. Furthermore, this equivalence test threshold distance is strikingly similar to the comparison approach discussed in section 4.1.2, above, that was used to compare mathematical models in physics to the results of experiment.

It is also worth noting how this statistic takes into account the problem of over-fitting. Given the data shown in Figure 4.3, a measure such as the Mean Squared Difference would only take into account the difference between the heights of the grey bars, which, as pictured, is very small. However, this similarity is actually an illusion: the *actual* means of these values are *somewhere in the confidence interval range*. The location of the grey bar merely indicates the *sample means*: the mean value of the subjects that happened to be observed, or the model runs that happened to be made. Basing a choice of what model is “good” on the sample means and ignoring the confidence intervals results in a model which is *highly tuned to match the particular group of subjects used*, rather than indicating anything about the population as a whole.

The equivalence test as described, however, has one of the same problems as the *t*-test. Everything described so far has been specific to measuring performance *means*. None of the formulation given above can be applied to comparing the variances between model and reality, nor any other statistic (mode, median, skew, kurtosis, etc.). The main difficulty here is that formulas for calculating confidence intervals either do not exist or are extremely complicated and sensitive to various assumptions about the distribution of the data. For that matter, all of the above calculations *assume the data points are normally distributed*. This assumption is certainly not always (or even usually) true for the sorts of behaviours typically investigated and modelled in cognitive science. What is needed, then, is a way to determine confidence intervals for *any* statistic, without making any assumptions about the underlying distribution of the data. If such a technique is available, then the equivalence testing approach can be used for any measure cognitive modellers may be interested in.

4.2.6 Bootstrap Confidence Intervals

The key step in the equivalence testing process is the determination of the confidence intervals. The technique for doing this described above assumes the data points are normally distributed, and is only applicable when comparing the means of the data. Since a good model should match the real-world behaviour in terms of its complete distribution, not merely the mean, and since many of the patterns that cognitive scientists are interested in are not normally distributed, some other approach is needed to determine the confidence intervals. This approach should be applicable to any desired statistic of the data distribution, so that it is possible to determine how well the variances, medians,

or skews of the model and real data match. It should also not make any assumptions as to the underlying distribution of the data.

Fortunately, there is one statistical tool that exactly meets these criteria: the *bootstrap method* for determining confidence intervals (Davison and Hinkley, 1997). The basic idea is simple, but computationally intensive, and so it is only with the development of modern computers that it has become feasible for use. The idea comes from the basic definition of the confidence interval as the range over which the measured statistic would vary if the experiment was repeated. As an example, consider an experiment with 16 subjects which results in the raw data shown in Table 4.3.

Raw Subject Data			
1	2	2	3
2	1	0	5
2	3	3	2
1	1	1	3

Table 4.3: Example data from a study with 16 subjects

For this particular sample of 16 individuals, a wide variety of statistics can be gathered.

A selection of these are given in Table 4.4.

Sample Mean: 2.0
Sample Median: 2
Sample Standard Deviation: 1.1726
Sample Skew: 0.69775
Sample Kurtosis: 3.5041

Table 4.4: Various statistics from the example data shown in Table 4.3.

However, these particular values are only the mean, median, and so on of the *particular group of 16 subjects in the study*. Researchers are not particularly interested in creating models of that particular group of 16 people; instead, the intent is to create a model of *the*

entire population. But, since the mean, median, and so on of the entire population is not exactly known (as the entire population cannot be sampled), the confidence interval is meant to indicate where this population value is likely to be. In the analytical technique used by the *t*-test, above, this is done by assuming that there is a normal distribution and then determining what would happen if the experiment were repeated an infinite number of times. The confidence intervals calculated above are the ranges encompassing 95% of those infinite repetitions of the experiment.

The approach taken in bootstrapping is to perform this repetition computationally, rather than analytically. That is, instead of relying on a mathematical manipulation which precisely determines what would happen if this repetition were repeated an infinitely large number of times, the bootstrap approach is to actually perform this repetition a large, but finite, number of times. The advantage here is that the standard statistical approach requires assumptions about the distribution of the data (i.e. a Gaussian curve), and is only mathematically tractable for simple statistics, such as the mean value.

Bootstrap confidence intervals make no assumptions about the underlying distribution, and can be determined for any desired statistic.

However, performing an computational repetition seems to be impossible. After all, the only available data values are from the original experiment. If the statistical measurements are simply repeated on the exact same group of 16 subjects, the same result will occur every time. It is also clearly not possible to actually bring in 16 new subjects and repeat the experiment thousands of times. Instead, the bootstrap approach is to approximate the effect of having 16 new subjects by *re-sampling the original set of*

subjects. That is, a new sample of 16 subjects is created by randomly choosing 16 data points from the original 16. This re-sampling is done *with replacement*, so the same data point may be chosen twice, and some may not appear at all in the newly created sample. What this means is that each new set of re-sampled data is representative of what would be *expected* if the experiment were repeated, given the observed variability in the original data. While the real measurements produced the original set of measurements, these re-sampled sets of measurements are unbiased representatives of what could have happened, and all represent values that would not be a surprise to observe.

For example, if the subject data values in Table 4.3 are resampled five times to produce five new data sets, the data shown in Table 4.5 may arise.

Resample 1	3	2	3	2	1	5	1	1	1	1	3	3	2	2	1	2
Resample 2	2	3	3	1	3	1	3	1	2	1	3	3	2	1	2	1
Resample 3	1	2	2	1	3	2	2	2	1	2	2	2	1	3	1	2
Resample 4	2	1	1	3	3	3	2	3	2	5	3	3	1	2	1	2
Resample 5	2	1	2	2	0	1	1	1	1	3	2	1	3	1	2	2

Table 4.5: Data from Table 4.3 resampled with replacement five times

None of these re-samplings are exactly the same as the original data. However, if the original measurements were repeated, each of these sets of measures could have occurred. These values thus represent a large set of plausible results, all of which are consistent with the original experiment.

The mean, median, and so on of each re-sampling is then calculated. The process is repeated over and over again, resulting in thousands of statistical measurements. The number of repetitions recommended is not well-defined, but 1000 to 3000 is sufficient for

all but the most pathological of cases (Davison and Hinkley, 1997). It is important to remember that, while this process may appear to be somewhat strange, it is actually exactly what is happening in the standard analytical approach to finding confidence intervals discussed in the previous section. The only difference here is that a finite number of re-samplings are occurring, rather than using mathematical analysis that requires assumptions about the distribution to directly determine what would happen with an infinite number of re-samplings.

Given this set of 1000 to 3000 values, determining the 95% confidence interval is simple. Each value represents a result that could reasonably be expected from the original observations. Thus, to determine a confidence interval, the values are sorted, and the top 2.5% and the bottom 2.5% are removed. The resulting range of values is the 95% confidence interval. Importantly, this process works for *any* statistical measurement, and does not involve any assumptions about the underlying distribution of the data, other than the minimal assumption that the original sample is representative of that population (i.e. there is no sample bias). For the data set shown above, the following 95% confidence intervals can be determined.

Mean:	1.4375 – 2.5625
Median:	1 – 3
Standard Deviation:	0.696 – 1.541
Skew:	-0.612 – 1.534
Kurtosis:	1.472 – 5.236

Table 4.6: 95% bootstrap confidence intervals for various statistics of the data shown in Table 4.3.

If this process is repeated for both the observed real-world data and the model data, then the equivalence test procedure described in the previous section can be readily applied.

The result is the maximum difference between the model and reality on *any* of these statistical measures of the distribution (with a confidence of 95%, or an error rate of 5%).

One complication with this process is the correction factor used in the non-bootstrap standard equivalence test. As described above, this is a shrinking of the confidence intervals based on the variance of the two data sets, and allows the standard form of the equivalence test to be more powerful, as it results in smaller ranges. However, this correction is based on two assumptions: that the observations follow a normal distribution, and that the measurement being made is a mean. In any other situation, this correction is not provably correct.

It would be possible to simply apply this correction regardless of its correctness when performing the bootstrap confidence intervals. This would have the effect of shrinking the confidence intervals by, in the best-case scenario, $1/\sqrt{2}$, down to approximately 70% of their original size. In a situation where the model and reality have highly different variances, this correction should would shrink the ranges by less than this amount. Tryon (2001) notes that if one standard deviation is five times larger than the other, the confidence intervals would be shrunk to only 85%. Thus one might be tempted to reduce the bootstrap confidence intervals by a similar amount, resulting in smaller difference values. Certainly, for measurements of the *mean* value, this might be a reasonable approach, if one is willing to assume normally distributed data. However, there is no mathematical justification for doing so for any other statistic. Indeed, if the intent is to be as conservative as possible (i.e. to make no unwarranted assumptions), then no correction factor at all should be applied.

The result is a single comparison measurement, based on bootstrap confidence intervals and the equivalence test. It produces a value that allows for the conclusion that any particular statistical measurement of model behaviour differs from the real-world behaviour by at most this value. Such a conclusion is guaranteed to be incorrect no more than 5% of the time, and works for any statistic on any sort of distribution.

While examples of this approach have not previously been found in the modelling literature, some of these ideas have been independently noted in a footnote in (Axtell et al, 1996, footnote 13). They note that a way must be found to reverse the standard null hypothesis test, and that a tool such as bootstrapping could be used to deal with the calculation of confidence levels for such non-standard distributions. However, when that paper was written the mathematics underlying the equivalence test were not well-studied, and no follow-up to this idea has occurred within the modelling field. This thesis will hopefully rectify this state of affairs by the above detailed description of the process, and the numerous examples of the use of this technique throughout chapters 7 to 13.

4.2.7 Multiple Measures

The bootstrap-based equivalence test described above works on a single statistic of a single measurement at a time. For example, it may provide a result indicating that the mean reaction time in a particular situation given by a model is within 50 milliseconds of the true mean reaction time in that situation. However, as was established in chapter 3, it is generally desirable to have the same model predict behaviour in a variety of situations. Furthermore, more aspects of the distribution than the mean performance of a model

should be considered. It is currently quite common for models to severely under-predict the variability in human data (Anderson & Lebiere, 1998), and a more complete modelling methodology should discourage this. It is thus generally the case that multiple measures will exist, and a successful model will be one that matches on as many of them as possible. These different measures could be any and all of the following:

1. Different statistics (means, medians, standard deviations, etc)
2. Different situations (condition A versus condition B)
3. Different types of measures (reaction times, error rates, learning rates, etc.)

4.2.7.1 Multiple Situations

The simplest and most common of these cases is that of different situations. Most studies within cognitive science involve varying some aspect of the task environment and determining how this affects some particular measurement. This is the standard paradigm with a control condition and an experimental condition. When modelling this situation, the model can be run in two situations: once in an environment that models the control condition, and once in an environment that models the experimental condition.

The result is two sets of values for the model, and two sets of values for the real subjects. The values in one condition can be compared using the equivalence test, and the values in the other condition can also be compared, but how should these two values be combined, so as to give an overall conclusion as to how good the model is overall?

One approach would be to avoid this problem by subtracting the values to determine an *effect size* and comparing the model and reality based solely on this value. The effect size is simply the difference between the subjects' performance in the two conditions,

indicating how much of a change in behaviour is caused by the change in condition. This sort of measurement is being identified as a suitable replacement for the standard null hypothesis testing approach, given the problems noted in section 4.2.4, above. If one is only interested in the model in terms of its ability to model the effect this condition has on behaviour, and is not in fact interested in the exact behaviour in either condition, then this approach could be suitable. However, it would be preferred to develop model which are accurate about all three aspects of the situation: the performance in condition A, the performance in condition B, and the difference in performance between A and B. Given this, the effect size measurement can be seen as yet another measurement that needs to be matched.

Another approach would be to average the equivalence test threshold values over the various conditions. If the model matches to within 70 milliseconds in condition A and to within 66 milliseconds in condition B, then the average value is 68 milliseconds. If the measurements under consideration are all of a similar type, but are in different conditions, then they will all be measured in the same units, and so this average can be determined. However, one problem with this approach is that it results in the same sort of obfuscation of the underlying data as was noted in the previous section about Correlation and Mean Squared Difference. By providing an average, information about how the range of variability of the data is lost. In this case, concluding that the “average” performance of the model is that it is accurate to within 68 milliseconds seems odd, since it does not distinguish this model (where the values are 70 and 66) and another model which is accurate to within 10 milliseconds in condition A, but only to within 126

milliseconds in condition B. Furthermore, taking an average opens up the possibility of artificially improving the performance of the model by adding more measurements *of the sort the model is particularly good at*. This hides the distinction between aspects of the situation that the model matches on well, and those that it does so poorly.

Instead, the most conservative approach to combining measures is to simply take the maximum equivalence test distance value of all the conditions. If a model is accurate to within 66 milliseconds in one condition and 70 milliseconds in the other condition, then the most conservative conclusion is that the model is, overall, accurate to within 70 milliseconds. For the case where the values are 10 and 126, the conclusion would be that it is, overall, accurate to within 126 milliseconds. Using this approach allows for accurate and clear overall conclusions, at the price of losing information about those conditions where the model is particularly accurate. However, if there are interesting patterns about what sorts of measures the model is good at, and what measures it is bad at, then *separate, more constrained conclusions can be made*. That is, the overall conclusion may be that a model is accurate to within 150 milliseconds, but in a particular sub-set of situations, the maximum error may be only 50 milliseconds. This sub-group of conditions can be identified and distinguished. By separating them this way, rather than combining the measures together into an average, useful distinctions are preserved, and the resulting conclusion does not imply that the model is more accurate than it is. It is this approach that will be used throughout the remainder of this thesis.

4.2.7.2 Multiple Types of Measures

It is unclear how to generalize this approach to other *types* of measures. If the experiment in question measures reaction time and error rate, how are these values to be combined? What is the maximum of 70 milliseconds and a 30% error rate? Furthermore, what is the maximum of a mean of 70 milliseconds and a standard deviation of 25 milliseconds? Since these are different sorts of measures, there is no clear way of interpreting what is meant by a combined most conservative conclusion. One could, of course, not combine them at all, resulting in a conclusion that the model gives reaction times accurate to within 70 milliseconds, gives a reaction time standard deviation accurate to within 25 milliseconds, and error rates accurate to within 30%. This may be the safest method for dealing with this combination problem.

However, if there are a large number of difference measures and a combined value is strongly desired, it is possible to *normalize* the error differences to a standard unit, and then find the maximum. This can be done based on the *relative size of the equivalence test difference as compared to the size of the real-world data's confidence interval*. That is, if the real-world information has a confidence interval range of 50 milliseconds, and the equivalence test indicates that the model is accurate to within 70 milliseconds, this could be seen as a *better* match than one where the equivalence test still indicates 70 milliseconds, but the real-world observations give a confidence interval that is only 20 milliseconds wide. The first situation is normalized to a relative difference of 1.4, while the second normalizes to a value of 3.5. This normalization can be applied to all the measurements, which can then be combined by finding the maximum relative difference.

If the resulting maximum value is, for example, 1.8, then the resulting conclusion is that the model is, on all of the measurements considered, accurate to within a range that is 1.8 times as wide as the range of the real-world data.

This normalizing approach is presented here as a possibility for combining measures. The more clear option is to resist combining, other than in situations where similar measures are being combined, and so the maximum value can be used. There may, however, be situations where a combined overall quality rating is desired, and so this normalized maximum difference approach may be used.

4.2.7.3 Statistics and Multiple Measures

One further issue must be considered when looking at multiple measures from the equivalence test. As it is currently defined, the accuracy of the equivalence test is based on confidence intervals calculated. If the 95% confidence intervals are used, then up to 5% of the time the model and reality may differ by *more* than the equivalence test concludes. This means that if 20 measures are examined, then on average one of the measures will differ between model and reality by more than is expected. One way to deal with this is to increase the confidence interval range to, for example, 99% (this approach is recommended by Tryon, 2001). This would decrease the occurrence of this problem at the expense of having larger differences.

Alternatively, one can highlight this fact in the conclusion itself. Each of the measures in the conclusion have a 5% chance of being larger than expected. For many of these measures, it will not matter, since the *maximum* is being used for the conclusion. Only a

few measures are likely to be close to this range, and they can be individually identified. For these measures, there are two possible sources of error: either the model is giving an inaccurate confidence interval (due to the model runs happening to have abnormally low or high values), or the real world is giving an inaccurate confidence interval (due to sample bias). The first of these possibilities can be investigated by re-running the model, producing more model data, and thereby recalculating a confidence interval. The second possibility is more difficult to address. However, it is an inherent problem in *all* science, and is a major reason why replication is an important part of the scientific method. If a model does not match reality, it is always possible that the real-world measurement was inaccurate, and this can be addressed by future studies.

4.3 Contributions: Measurement in Computational Cognitive Modelling

The previous sections have discussed measurement from the point of view of physics, and have discussed a variety of different existing measurement techniques and their advantages and disadvantages. In this section, these approaches are developed into recommendations for computational modelling in cognitive science. The result of this analysis will be a novel set of guidelines for performing modelling work in cognitive science. Developing and justifying this methodology is the core contribution of this thesis.

4.3.1 The Relativized Equivalence Measure

In section 4.2, various methods were discussed to compare a given model to reality. The equivalence test with particular additions for combining multiple measures was argued to be the best, as it provides the sort of conclusions that are desired, and are similar to those found in standard physics modelling. This measurement technique is a core contribution of this thesis, and is the basis for the modelling methodology developed herein. As such, it is useful to have a precise algorithmic specification of the measure.

The main basis of the measure is the basic comparison of model and reality introduced in section 4.2.5. Here, the maximum difference is found between the model's confidence interval and the empirical data's confidence interval. This indicates how equivalent the model is known to be to the real system, and so is called the equivalence (E). If the model confidence interval is M_L to M_U and the real-world confidence interval is R_L and R_U , this formula can be written as follows:

$$E = \max(M_U - R_L, R_U - M_L)$$

This basic formula only deals with a single measure. If there are multiple measures to be considered (as there usually should be), separate equivalence values can be determined for each one. This is reflected in the following formula, where the index i indicates the different measures.

$$E_i = \max(M_{i,U} - R_{i,L}, R_{i,U} - M_{i,L})$$

Since these measures may be in different units, they must be relativized before they can

be combined. This is done by dividing each measure (E_i) by the amount of difference that is considered acceptable. This amount may change based on the requirements of the model: in some situations, it may be sufficient to look for models that provide reaction times within 50 milliseconds, for example, or an error rate within 10% of the human performance. This scaling factor is thus a subjective judgement about how close is “close enough” on each of the measures.

However, there is an objective lower bound on the size of this scaling factor. Since the size of the real-world confidence interval is known, there is no reason to expect a model to be *better* than this range. That is, the scaling factor should never be smaller than $R_U - R_L$. Using a value less than this would attempt to find models which are more accurate than the can be supported by the available empirical data. Given this lower bound, the relativized equivalence measures are as follows:

$$E_i = \frac{\max(M_{i,U} - R_{i,L}, R_{i,U} - M_{i,L})}{R_{i,U} - R_{i,L}}$$

As argued in section 4.2.7, these individual measures can be combined together into one overall equivalence measure that reflects the model's worst-case performance on all measures under consideration. This involves simply taking the maximum of all of the available E_i values.

$$E_r = \max_i \frac{\max(M_{i,U} - R_{i,L}, R_{i,U} - M_{i,L})}{R_{i,U} - R_{i,L}}$$

This measure is referred to as the *Relativized Equivalence Measure*, and is denoted E_r . It

is the main measure used throughout the rest of this thesis and in the computational cognitive modelling methodology being proposed. If bootstrap confidence intervals are used, it can be applied to any statistical measure used in any empirical situation, and makes no assumptions about the underlying distribution of the data. It is a generic measure of how close the model and reality are guaranteed to be, within a particular level of confidence.²¹

The relativized equivalence measure is highly conservative, but makes very strong claims about the relationship between the model and reality. A value below 1 indicates that every one of the individual measures used is within the bounds of equivalence with 95% confidence. This means that the model is good for every measure, rather than just being good on average over the measures. However, if one model has an E_r of 0.9 and another has an E_r of 0.8, this does not mean that the second model is better than the first. Rather, both models are equally plausible explanations of the cognitive phenomenon. If a researcher desires to distinguish between these models, then new empirical observations are needed.

4.3.2 Parameters, Explanations, and Prediction

When creating computational cognitive models, it is very often the case that the underlying theory does not fully specify everything about the model. For example, a theory might specify that if a reward occurs, the association between a stimulus and the

²¹ This way of describing relativized equivalence is to a certain degree similar to the Probably Approximately Correct approach from Machine Learning (Valiant, 1984). However, in PAC, the probability refers to whether a particular learning algorithm will, given a pattern of training data, result in a decision rule which is accurate to within a given range of error. With equivalence learning, the 95% probability refers to the uncertainty in what the model is being compared to (and the performance of the model itself).

response should be increased. However, the exact *numerical size* of this increase may be unspecified. When this theory is turned into a computational model, this becomes a *parameter* that must be set to a particular value before the model can be run. Setting this parameter to different values will, most often, produce different results. This difference may be small and inconsequential, or it may be very large.

Since varying the parameter value will, in general, affect the performance of the model in unpredictable ways, each of the different parameter settings can be said to result in a *different model*. These models will be very similar in terms of their algorithm and structure, but the different parameter values may strongly influence the overall behaviour of these models. Referring to them as different models (as opposed to one model with different parameter settings) is meant to highlight this possibility. Understanding what a model does with one set of parameters does not necessarily indicate what it will do with another set of parameters.

This fact that there will generally be multiple models which vary due to parameter settings has important consequences for the interpretation of the results of modelling work. To demonstrate this, consider an attempt to model an extremely simple system: the flipping of a coin.

To gather the real-world data to compare to, the coin is flipped multiple times. In this case, the thirty results are indicated in Table 4.7, with a 1 indicating Heads and a 0 indicating Tails.

1	0	1	1	0	0	1	0	1	1
1	0	1	1	1	1	0	1	0	1
1	0	1	0	1	0	1	0	0	1

Table 4.7: Example data from flipping a coin 30 times. 0 indicates tails and 1 indicates heads.

The confidence interval for the mean of these data points can be calculated using the bootstrap approach described in section 4.2.6. The sample mean is 0.6 and the confidence interval for this mean is 0.4 to 0.833. This means that the actual average for the 30 samples is 0.6, but the population average (i.e. the mean that would be found if the experiment were repeated many many times) can only be confidently said to be between 0.4 and 0.833. It should be noted that this range does include the actual population average for flipping a coin, which is 0.5.

If one were to attempt to model this system without knowing the actual population mean (which is only known in this case because it is such a simple system), one might be tempted to create a computational model for it. There would be no inputs, and the output would be wither a 0 or a 1. A simple model might look like the following

- | |
|---|
| <ul style="list-style-type: none"> • Output a 1 with probability p • Otherwise, output a 0 |
|---|

Table 4.8: A very simple model of coin flipping, with a single parameter p .

Since the parameter p can take on many values, there will be one version of this model with $p=0.1$, another with $p=0.2$, and so on. In theory there could be an infinite number of the models, and it is impossible to try them all. In this case, only the following 11 models are to be considered.

Parameter	Values										
p	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Table 4.9: Parameter values to be considered for the coin-flipping model

Running each of these different models multiple times results in the data and confidence intervals shown in Figure 4.4.

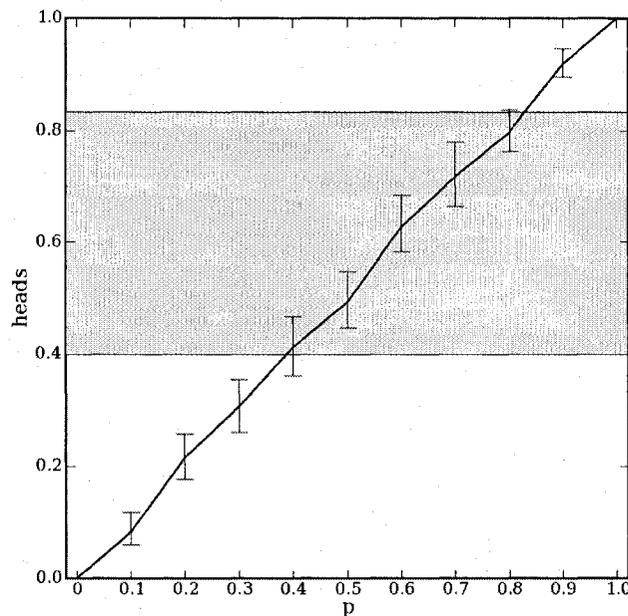


Figure 4.4: Eleven models with different values for parameter p compared to the experimental data from Table 4.9. The experimental confidence interval is shown as the shaded area.

To interpret these results, the equivalence test measure from section 4.2.5 can be used.

Here, the maximum possible difference between each model's confidence interval and the real-world confidence interval is found. This is shown in Figure 4.5. This graph shows, for each model, the maximum difference between it and the real world that can be expected. For example, the model with a p value of 1.0 could differ from the real world

performance by up to 0.6. The model with a p value of 0.6 only differs from the real world by at most 0.3. The shaded area on the graph is the size of the real-world confidence interval (i.e. the shaded area in Figure 4.4), for comparison.

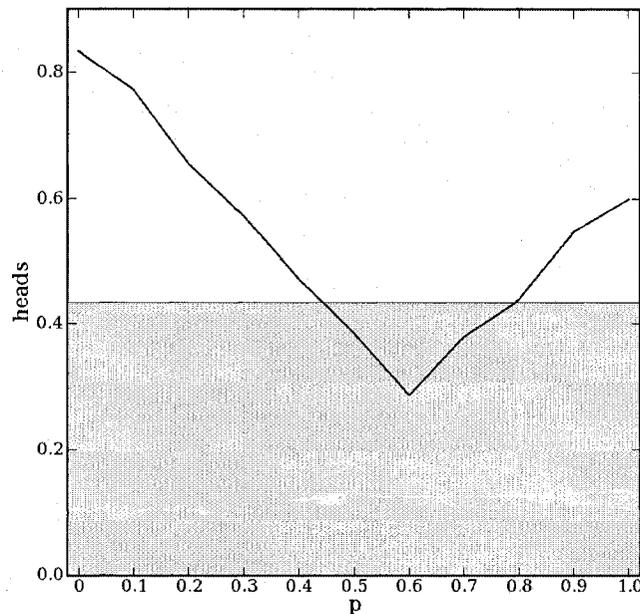


Figure 4.5: Equivalence test measure for each of the models shown in Figure 4.4.

One way to interpret this is to say that the best model for the real-world coin-flipping task is the model with $p=0.6$. This is the model that most closely matches the real-world performance. This is the sort of statement that is often found in computational modelling research, as a particular parameter setting is found to “best fit” the experimental data.

It is certainly true that the model with a p value of 0.6 gives the best behavioural match to the experimental data. However, this conclusion gives cause for concern. Since this is such a simple situation, it is known that the actual real best model is the one with a p value of 0.5. The 0.6 model only happened to work out to be the best due to the random

sampling error that occurred when collecting the original experimental data. In other words, the 0.6 model is, in some sense, *over-fit* to this particular situation.

What is desired is a way to identify the set of models which *could* be reasonably expected to match the real-world behaviour, taking into account this sampling problem. In this case, such a technique should identify the model with $p=0.5$ as a potentially good model, but not the $p=0.1$ model, for example. How this is done is quite simple, and derives directly from the confidence interval calculations. *Any data point inside the shaded area should be considered a potentially good model.*

That is, every model whose equivalence measure is less than the size of the real-world confidence interval could potentially be the perfect model for the system. Indeed, since the actual real-world mean is in general *equally likely* to be *anywhere* inside the confidence interval, every one of the models in this range has the same chance of being the right model. This point bears repeating: the above graph *does not* indicate that the 0.6 model is “more likely” to be true than the 0.5 model or the 0.7 model. There is no statistically legitimate means of distinguishing these three models without further empirical data on the situation.

Because of this, emphasizing the one model which happened to be numerically closest to the sample data (in this case, the 0.6 model) is not something that should be done in modelling work. There are certainly situations where that one model might be useful, however. For example, if an accurate prediction was required, and the only information available was that provided so far, then the model with a p value of 0.6 would be the best

one to use, as it is known to be accurate to within 0.3. However if the modelling work is part of a larger process to explain some cognitive phenomenon, then every model within the shaded area must be treated equally.

Overall, the model with the smallest equivalence test measure may be the best predictor, but any model which differs from the real world by less than the size of the real world confidence interval must be considered when trying to explain.

4.3.3 Parameter Fitting

In the ideal and simplest case, there would be no *free parameters* in a model (that is, no parameters for which the theory does not specify values). A model can become a zero-parameter model if the theory specifies particular values for these parameters. Indeed, this process can be seen as one of refining the theory to become quantitative rather than qualitative, and is one of the major benefits computational modelling provides. However, it is also possible that a theory requires different parameter values *in order to fit different people*. After all, a cognitive model can be a model of *a particular individual*, rather than a model of the group mean behaviour.

Unfortunately, this results in a situation where *parameter fitting* is a common occurrence in computational modelling research. A model is created, and then the parameters are adjusted until the “best” values are found (i.e. the values which result in behaviour which most closely matches that observed in the subject or subjects). The quality of the model is often argued to be related to the quality of this fit (i.e. if model A can be made to fit more closely than model B, then model A is deemed to be “better”.) However, as has

been correctly pointed out (for example, Roberts & Pashler, 2000), demonstrating that a model can be adjusted to fit a particular set of data *does not in itself inform you about the validity of the theory*. In particular, it says nothing about what range of real-world data the model could have been adjusted to fit. Perhaps by adjusting parameter settings it would be possible to match the model to *any* plausible set of data. If this is the case, then demonstrating a good fit merely indicates that the model is highly adjustable, not that it captures some important aspect of the particular situation being modelled.

Theorists who use good fits as evidence seem to reason as follows: If our theory is correct, it will be able to fit the data; our theory fits the data; therefore it is more likely that our theory is correct. However, if a theory does not constrain possible outcomes, the fit is meaningless. (Roberts & Pashler, 2000, p. 359)

The argument here is based on the idea that if a model is flexible enough and has enough free parameters, then it can be “tweaked” to match any possible observed result. A theory which allows this sort of flexibility is empty and unfalsifiable. However, this is only the case if parameter tweaking can, in fact, achieve *any* possible (or even just plausible) result. In practice, computational models, especially those based on the algorithmic, physical, and behavioural considerations described in the previous section, do not tend to be infinitely flexible. The possible ranges of behaviours covered by the theory can thus be examined and described as part of the modelling process.

4.3.4 Predictive Fitting

One of the key recommendations presented by Roberts & Pashler (2000) is to highlight the predictive aspect of modelling. This point is also raised by Pylyshyn:

“To be explanatory, a theory cannot have as many free parameters as there are data points. Put another way, we would not be content with a theory that must be changed each time a new observation is made even if, at any given time, the theory accounts for all available observations (that is why the *pre* appears in *prediction*).” (Pylyshyn, 1984, p. 122)

The idea here is that even if a model needs to be tuned to suit particular subjects, it should still be evaluated in terms of its predictions. Clearly, these predictions need to be about aspects of the subject's behaviour *other than those considered for the tuning*. That is, a sub-set of the observed data can be used to determine particular values for various parameters for this subject, and then the resulting model can be evaluated based on the rest of the observed data.

However, one must be careful when selecting the sub-set of data for fitting. As a simple example, consider the observed data of reaction times in conditions of low-stress, medium-stress, and high-stress in the following table:

Stress Level	Reaction Time
Low	500 msec
Medium	1000 msec
High	1500 msec

Table 4.10: Sample Data for Predictive Fitting

If one was to use the Low and High stress data as the basis of parameter fitting, and then use the resulting model to predict the Medium stress value, this would not be an impressive prediction. This is because this sort of prediction can be done merely by a linear interpolation of the data used for fitting. In other words, predicting observed behaviours that are *similar* to those conditions used to fit the model is not an impressive test of the model. If a simple linear model (i.e. one that does not postulate any internal

structure, but is merely tuned to fit the observed data) is capable of producing equally good predictions, then a more complex model is not useful. In Kaplan, Miller, & Carley (1996), this comparison to a linear model is referred to as *harmonizing*. The linear model is meant to serve as a benchmark for comparison, indicating the capabilities of a completely naïve model.

However, a more convincing predictive argument can be made if the predictions *are in a different domain* from the data used for fitting. That is, if the behavioural data used to fit the model and the data used for comparison are of different *types* (such as the rates of different sorts of errors, or reaction times), then the model must be capturing an underlying commonality between these situations (one that would be impossible for a naïve linear model to arrive at). A fit using reaction time data that resulted in accurate error rate predictions, or a fit using error rates that predicted the results of lesion studies would be preferred over results involving merely one of these domains. This can be most clearly seen in recent work with the ACT-R models²² where reaction time fits are used to predict fMRI results (Anderson et al, 2003).

Just as the gravitational theory was useful for the first 120 years before the parameter G was known, so too can theories based on computational models which do not specify the parameters of their models. In these situations, the application of the theory requires some process whereby the parameter can be determined for the situation in question. Once this process, which usually involves using the model in some particular situation and finding the best-fitting parameter setting, is complete, the model can then be used to

²² The ACT-R cognitive architecture is discussed in more detail in chapter 11.

predict future aspects of the behaviour of that particular cognitive agent in that particular situation. That is, we perform parameter-fitting to create a model of this special case, and then can use that model to perform predictions. Importantly, it is this second stage which is the test of the theory. Merely finding a parameter setting which fits indicates the flexibility of the model, and certainly could lead to the establishment of a model which is behaviourally equivalent on the particular behaviours measured. However, if the model can then be shown to generalize to new behaviours, then the result is a stronger model.

4.3.5 Parameter Ranges

In section 4.3.1, above, the effects of changing the parameter p within a coin-flipping model was examined. Although the most numerically predictive model was found to be one with a p value of 0.6 (due to the particular data sample being used), the overall result was that the model produced data statistically indistinguishable from the real world data for any parameter value between 0.4 and 0.8. That is, given the available evidence, all models within that range must be considered to be equal.

As a particular theory develops, and more data points are analyzed, and more situations are considered, the parameters within it may become more completely specified. That is, it may be found that the parameter should always be set to 0.5, for example. In this situation, that particular parameter setting can be considered now to be *part of the theory itself*, and no longer freely adjustable. These become *canonical* parameter values (Anderson & Lebiere, 1998), and are an explicit goal of many modellers.

However, a more likely scenario is that a *range* of parameter settings will continue to be

found to be suitable. In this case, the information gained by fitting the model to a variety of situations can indicate a constrained set of values the parameter can take on.

Importantly, there are two different ways in which this parameter range constraint can occur.

The simplest type of parameter ranges arise in situations where a parameter may have to be a particular value in one situation, and a different value in another situation (in order for the model to produce accurate behaviour). If these two different situations involve different subjects, then this parameter may be interpreted as something to do with the individual differences, and a similar difference with the same parameter may be found in other situations as the model is expanded. If the two values involve the same subject but in differing experimental situations, then the goal will be to identify aspects of that situation that result in the parametric difference. These are typical processes followed in computational modelling research.

However, there is a second type of parameter range specification which has not received as much attention. It is somewhat addressed by *sensitivity analysis*, where the results of varying parameters across a wide range are determined. This involves running a model using many different parameter settings and determining how well each of these different models performs. Roberts and Pashler (2000) recommend this to indicate the full range of data that the model *could* have fit, and argue that if this is a wide range, then merely fitting the data is insufficient (as discussed above). In other words, if there is a very large range of possible behaviours that a model could produce, merely by adjusting its parameters, then it is unsurprising that it can match closely to the observed behaviour.

The matching is then more a measure of the flexibility of the model, not anything to do with its accuracy. This line of reasoning has lead modellers to include specific constraints, such as specifying particular canonical parameter values for the models (e.g. Anderson, 1998).

However, this response of focusing on choosing particular specific parameter misses a key underlying issue. The more important aspect of this situation is discovering that a parameter's value can vary across a whole range *without affecting the accuracy of the predictions*. That is, a model's behaviour may be *statistically indistinguishable* no matter where (within the given range) the parameter is set. In this case, it is inaccurate to indicate one particular “best” parameter setting within that range, even though there may be one setting which is marginally closer to the observed behaviour, due to random chance. This sort of conclusion would be an *over-fitting* of the observed data. Given the statistical variability of the behavioural data considered by most modelling work, this should be a common phenomenon.

If ranges of parameter values are considered, then modelling moves from concluding “this model is matches if parameter X is set to some unknown value between 0 and 1” to concluding “this model is good if parameter X is set to *any* value between 0 and 1”. This would, of course, use the bootstrap equivalence test measurement described above. That is, as some parameter of the model is varied, the equivalence test threshold can be measured for each model. If these values are graphed, a curve such as that seen in Figure 4.5, above, is generated. This characteristic U-shaped curve indicates that all of the models in the middle are approximately equally good models (they all have about the

same maximum worst-case difference from reality), but the models with low or high values for the parameter are likely to be much worse models.

Without this analysis, it is possible to have a model with very precise numerical predictions and very precise parameter settings which give a *false sense of accuracy*. A theory may claim that a particular parameter should be 1.0, when in fact it is the case that it would be just as accurate at a value of 0.1 or 10. A required part of testing a theory is then to show that the theory would *fail to predict accurately* if the parameters it specifies were outside of some range.²³ That is, instead of finding a particular best-fit value, a range of settings can be found, all of which produce models that are equally good. As more data points are gathered, this range may shrink to become more precise, or split apart for individual and situational differences. However, it is insufficient to indicate that a particular value gives a good model without also showing that other values do not.

4.3.6 Non-numerical Parameters

To further complicate the issue of parameter variation, *every aspect of a model can be considered a parameter*. That is, there are always any number of changes that could be made to a model. Only some of these are the sort of numerical values that have been considered thus far as parameters. However, any of these other possibilities can also been seen as parameters. Representational choices such as local or distributed representations, algorithmic choices such as what learning rule to use, and even structural choices underlying the theory as a whole can all be considered non-numeric parameters

²³ It is also possible that the model would give accurate predictions for some complicated set of parameter values, such as “any prime number”, or “any value between 2-10 and 400-410”. However, there will almost always be some range of values around any one suitable value which can be identified.

of the model. As such, they should be subjected to the same analysis as the numeric parameters. Models must be tested with multiple different choices for various aspects of the theory, so as to determine the range of theory choices that are compatible with the observed real-world behaviour. It should not be the case that only the aspects of a theory that are amenable to numerical representation are investigated.

In the RELACS model discussed above, a variety of aspects of the model can be seen as non-numerical parameters. There are many possibilities for the formulas used to determine R_j and W_j . There are many ways to adjust the recall of previous events in the second decision rule. There are many ways of choosing which of the three rules to use. There are many more rules which could be added. All of these possibilities can be thought of as non-numerical parameters, and need to be investigated. Any research which only examines the particular parameters which appear as numerical values in the various formulae in the model is artificially limiting the scope of inquiry.

An interesting practical example of this problem appears in the work of Sibley and Kello (2004). They show that adjusting the slope of the neural activation function (a numerical parameter) in a particular neural network implementation of a word-pronunciation task resulted in a high-level change to the behaviour of the system. In particular, they showed that adjusting this parameter was equivalent to adjusting the model from using localist representations to using distributed representations. That is, a numerical parameter was *acting as if* it were choosing between two alternate ways of implementing the model. Sibley and Kello describe how this adjustment generates the same sort of behavioural data as seen in double dissociation tasks, calling into question the standard conclusion

that a double dissociation indicates the involvement of two separate systems interacting.

In other words, there is no sharp distinction between numeric and non-numeric parameters. Something as complicated as having two different ways of representing data within a model can turn out to actually be controlled by a single numeric parameter.

Importantly, this can happen by accident, without the creator of the model intending this to be the case.

This means that there is a continuum of variation: numerical parameters, alternate components, and even entirely different architectures. Determining the flexibility and bounds on a model is not a matter of sampling values in a constrained, finite-dimensional search space. Instead, it is a matter of searching through a space of different models, some of which may only differ by a particular parameter value, while others may have completely different implementation choices. The goal is to identify a region in this “model space” which contains models equivalently comparable to the real system.

Unfortunately, this is a more difficult situation to deal with, as there is no way of systematically trying every possible alternate version of a particular sub-component, or even every possible one that is in some sense “close to” a given version. The space of possible versions is not one-dimensional. Certainly, a number of different versions can be tried, and some will result in accurate predictions, and some will not. Describing this range of suitable versions (and which versions are not suitable) *is as important as specifying what numerical parameter values are suitable and not suitable*. If only one structure is tested, then the same error is being made as if only one particular numerical

parameter value is considered.

4.3.7 Model Comparisons

Since all models (including ones with zero free numerical parameters) will have some sort of variation possible in their parameters or structure, modelling analysis must necessarily involve comparisons of multiple models. This allows us to avoid overconfidence in a particular model, when many other equally plausible models may be just as suitable.

[O]ne might propose only one among many possible mechanisms and have no warrant for thinking it is the one actually utilized. This is hardly better than having no explanation at all. (Bechtel & Richardson, 1993, p.21)

Roberts and Pashler (2000) also point out this problem, indicating that the same data can be fit by similarly complex theories which make different assumptions. To address this, we need to not only examine what behaviours the model could be fit to (as discussed above), but also what *models* could fit the observed behaviour, and what models cannot.²⁴ Studying a single model (or even a set of models all based on the same architecture, with numerical parameter differences) can lead to a false sense of the model's accuracy. The context of other models being compared is also required.

The actual internal details of the models specified by a cognitive theory can be *any* algorithmic method. The only constraint is that the model allow for the definition of certain values to constrain the model for a given situation, and for the determination of certain values which are to be compared to the real-world measurements. This covers a

²⁴ This process is markedly similar to Bacon's scientific method involving testing a theory by also testing a competing and incompatible theory.

vast variety of possible systems. Indeed, one of the major reasons for using computational models in the first place is that they allow us more flexibility than is found in mathematical models, which must be simple if they are to remain tractable.

It is thus expected that *multiple* potential models (generated by different theories and different parameters) will exist. As these models are developed, they may match to different ranges of situations, indicating different domains of utility for those theories.

This is a typical situation from mathematical models in physics, as seen in the development of different theories of gravity, or the planetary models of Ptolemy and Copernicus. The initial, simpler, more restrictive models are a necessary first step toward the development of the broader models, and provide points of comparison.

4.4 Summary

4.4.1 State of the Art

- In physics, comparisons between model and reality are done by determining how different the two could be. This places an upper bound on how much error there could be.
- When comparing model performance with real world performance, one should ensure that the *distribution* of behaviour is the same. That is, it is not enough to just indicate that the mean performance is the same. Other aspects of the distribution should also be equivalence (such as the standard deviation, kurtosis, median, etc.).

4.4.2 Background

- There are many statistical techniques used to compare models to reality. Most rely on looking at the difference between the mean model performance and the mean real world performance (correlation and mean squared difference). This has the problem of ignoring the confidence intervals, which makes it possible to over-fit the sample data.
- Correlation can be highly misleading, as the numerical value is only sensitive to the large-scale features of the data. Mean squared difference only considers how much the model is in error *on average*, and thus can hide the fact that some measures are much worse (or better) than others. The t-test and the χ^2 test can only disprove a model; if the test is passed, this is merely evidence that the model is not contradicted by the available data.
- The equivalence test is a measure which determines an upper bound on the amount of difference between two sets of data. It involves the probability of observing a given sort of data if the means of the sets of data differ by more than a certain amount. If this value is less than 0.05, it can be interpreted as indicating there is a 95% confidence that the means do not differ by more than that amount.
- Bootstrap confidence intervals generalize the standard approach to determining confidence intervals. Instead of assuming that the observations come from a normal distribution, it uses a computational re-sampling technique to generate a large set of data values that *could have* been observed. This set provides the basis

for finding confidence intervals for any desired statistic, such as the mean, variance, median, and so on.

4.4.3 Contributions

- The equivalence test was combined with the bootstrap confidence intervals, resulting in a simple equivalence measure that indicates how different the model and reality could be (with 95% confidence). This measure makes no assumptions about the underlying distribution of the data, and is suitable for any statistic (standard deviation, kurtosis, median, etc.).
- It was shown that averaging over multiple measures leads to important information about the limitations of a model being lost. Instead, an approach was presented that combines multiple equivalence measures by taking the maximum value. This is only appropriate if the measures are all of similar types (e.g. all measures of accuracy, or all measures of reaction time). To combine different sorts of measures, the equivalence must be scaled to a similar range. This scaling factor should be the amount of difference between model and reality that is considered acceptable. This can be no smaller than the real world data's confidence interval (as the model can never be expected to be more accurate than is warranted by the available data).
- It was shown that analysis of models generally results in a set of parameter values all of which produce results that are equally good, using the equivalence measure described above. This means that all models that are equivalent can be treated as

equally likely models. This is true even though one of those models will be the closest numerical match. The one that is the closest numerical match is over-fit to that particular sample, and may be the best choice for *prediction*, it was argued, but all of the equivalent models are equally good *explanations*.

- Both numerical and non-numerical parameters were shown to be potentially equivalent. This means that standard approaches which use a different strategy to deal with structural changes to the model than they do to deal with numerical changes may not be appropriate. Instead, these different sorts of changes can be treated similarly.

5 Developing a Methodology: Communication

The previous two chapters dealt with issues involved in how a cognitive researcher should go about creating and investigating computational models. What has not yet been dealt with is how this researcher can be communicated to other researchers. Scientific inquiry is a group process, and it is a general requirement that research be able to be transmitted to and continued by other members of the scientific community. This chapter thus deals with the rather different question of how models can be designed to be used by many different researchers, how the results of modelling simulations can be summarized and communicated, and how modellers can build upon the work of others.

5.1 Communication in Physics

In mathematical modelling, a complete representation of the model is generally presented when the model is introduced. The models are specified using the language of mathematics. This same language is also used to define how the models are meant to be used (i.e. how to convert a real-world situation into a model, and how to perform the statistical analysis of the resulting predictions). Having a complete description of the theory in question is a basic assumption in physics, and no discussion of a theory can take place without it.

However, this process is assisted by the *simplicity* of theories in physics. They are generally completely describable in a few lines (certainly less than a page) of mathematical formulation. Given this formulation, mathematical manipulations are used to determine the implications of the theories. This can a difficult process, and Hersh

(1997) identifies numerous errors in mathematical proofs that have been highly influential, many involving the misinterpretation and misuse of mathematical approaches inapplicable to the situation. However, by encouraging clear communication of exactly what is being put forth in a theory, and by allowing any researcher full access to that theory and the situations it is being applied to, progress is certainly made.

5.2 Replication and Communication

Since the implemented computational models can be thought of as precise specifications of the underlying theory, it follows that the *source code* itself is the most complete method for communicating about the model.²⁵ While the high-level verbal description of a theory is necessary for forming a quick overall impression, the exact details present in the computer code are necessary for interpreting any possible ambiguities that can arise. Furthermore, having the source code available also facilitates the *use* of the model by other researchers.

Historically, the modelling community has focused on publishing verbal (and pseudo-code) descriptions of their models. The intent has been that the published descriptions detail the core vital aspects of the model, and any further aspects in the source code can be considered mere implementational details. However, as has been pointed out repeatedly (Axtell et al, 1995, Axelrod, 1997) and as I have personally observed numerous times,²⁶ the description of the computational model provided in the publication

25 Complete pseudo-code for the model could also be used to give a complete description of a model. However, proving that the pseudo-code is complete and accurately represents the implemented model is a difficult task.

26 This problem is prevalent enough that when I teach a graduate level computational modelling course, the final project is to attempt to replicate a published modelling paper, identifying aspects where insufficient information is provided or where the provided information is inaccurate. Students tend to

is generally an incomplete specification of the model. This is a natural consequence of the complexity of models; a journal paper is unlikely to provide space for a complete description of the model, and indeed any prose description of the source code has plenty of potential for error.

In (Axelrod, 1997), eight core social science models (generally simpler than most cognitive science models) were the targets of attempted replication. Numerous problems were encountered, including “ambiguity in the published descriptions, ... gaps in the published descriptions, ... [and] situations in which the published description was clear, but wrong.” Fortunately, none of these problems turned out to invalidate the original results. However, given the complexity of models, this phenomenon should be expected, and in response it is becoming more common for modelling results to be accompanied by published source code.

Indeed, it is odd that computational models are not commonly shared among researchers, but this state of affairs is common to research involving computer models. In the field of Artificial Life, which focuses on the creation of computational models of animal-like “animats”, the same problem occurs.

Sharing work has been so difficult that researchers tend to build their own animat minds and worlds from scratch, often duplicating work that has been done elsewhere. There have been a number of attempts to re-use animat or agent minds and worlds, but the model of re-use often requires installation, or even a particular programming language. ... Often, the only person who ever does experiments with an animat or agent is its author. In this field it has become acceptable not to have direct access to many of the major systems under discussion. (Humphrys & O'Leary, 2002)

find numerous issues, and my own publications are not immune to this problem.

This state of affairs is clearly undesirable for a scientific endeavour.

5.3 SHRDLU: A Cautionary Tale

To further emphasize the importance of this step, it is worthwhile examining SHRDLU (Winograd, 1971), one of the great initial successes of the field of Artificial Intelligence, but which has not been successfully run since 1974. Available source code is incomplete, buggy, and incapable of producing the outputs shown in the original published accounts.

The original system was capable of allowing a user to interact with a simple “blocks world”, and able to interpret commands such as “FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU ARE HOLDING AND PUT IT INTO THE BOX.” However, the version of the program capable of performing this activity was produced by “direct patches to the Lisp assembly code and [Winograd] never back propagated them to his Lisp source.” (Semaphore Corp, 2005) In other words, the compiled version of the program was modified to produce the observed behaviour, but these modifications were never put into the original source code. This, combined with the multiple differences between modern versions of Lisp and the version the original program was written in, has meant that no working copy has existed since 1974. A reimplemention based on this code was attempted in 1999 at <http://web.umr.edu/~shrdu/>. The resulting version “isn't capable of completely reproducing the classic demo dialog and is fairly brittle and easily crashable, but it does correctly handle a large portion of the classic input sentences and many reasonable variations.” (ibid)

Certainly, many of the problems encountered by the SHRDLU project would not be

applicable for modern computational models. Programming languages change much less quickly now, and backwards compatibility is a higher priority. It is also exceedingly uncommon for direct changes to be made to the compiled object code, as opposed to being made to the human-readable source code. The key point, however, is that not ensuring the accessibility of the original and complete specification of a model is likely to negatively impact the usefulness and understanding of that model within the wider research community. This means that without this complete specification, there is no way to know exactly what the theory is that is being investigated. That is, any research which does not provide a complete description of the model(s) being researched is of questionable use to the overall scientific community.

5.4 Complete Specification

Furthermore, it is important to note that more than just the source code for the model is required. Merely having a runnable version of the model of the agent does not allow one to attain full replication of the results of the model. This requires the source code *for the environment the model interacts with*. Just as the real-world subjects are put into particular experimental situations in order to measure their behavioural responses, the computational model of the same situation also requires a *model of the environment* for the same interaction to occur. This is less necessary for simple, easily described environments, but should be considered part of the full modelling process.

Since the goal here is to allow both the reuse of the model, and the replication of the published results, a complete description of the experimental stimuli is required. This

can include such aspects as the complete set of stimuli, the representational format for the stimuli, the presentation order, any dynamic interactive properties of the environment and so on. In psychology, such a complete description of an experiment is explicitly desired and attempted (albeit impossible to achieve, given the complexities of real-world environments). However, in computational modelling, such a complete description is already available, as these details are all present in the computer code that was actually run. This makes a *complete* communication about the model feasible.

Taking this idea one step further, Lane and Gobet (2003) discuss the generation of automated test suites which can replicate the core important findings for a model, and which can be automatically run when one is making modifications to a model so as to confirm that the new model still has the important features of the old one. Furthermore, these tests can be run to compare multiple models to each other *using exactly the same tests*, with no extra effort on the part of the researcher. Making this system available allows for the seldom-attained goal in science of perfect replication. By having the complete source code for a model and the complete set of tests and stimuli used, anyone, anywhere *can exactly replicate* all of the data produced in a computational modelling experiment. This is especially important, given the fact that there are more models out there than there are current comparisons between models (Guilot and Meyer, 2000). By allowing for this complete replication, other researchers can perform comparisons beyond those considered by the original researchers developing the models.

Concerns have been raised that such a complete publication of a model is asking too much of modellers. Such source code could be considered proprietary, as could the

experimental stimuli being used, or the raw real-world data needed for comparison.

Humphrys and O'Leary (2002) argue that people who develop models should set their software up to run on a publicly accessible computer, so that other researchers can log in remotely and use the system. In this way others could make use of the system without actually having access to the proprietary aspects of it. This approach is somewhat cumbersome, and may be considered to be antithetical to scientific progress (as it hides the very models being investigated), but in any case it is not a route that has achieved wide-spread popularity.

It is also true that modellers are often hesitant to make source code to models available due to them being 'messy' or 'unreadable'. Some would argue that this means more time should be spent on organizing source code before publishing (Lane & Gobet, 2003).

However, in lieu of such a time-intensive process, it may be better to just make the code available anyway, in whatever state it is in. After all, given the experimental nature of model development, it is quite likely that the first version of a new modelling system will not be highly readable and well-organized, and perhaps it even should not be, as any newly created model is bound to involve numerous changes and re-thinkings. Indeed, it may be that it is better for *someone else* to re-implement the model into a clean and more widely useful format (possibly even in a different language). Having the original code available, however untidy it may be, allows for a reimplementations that can be directly tested against the original.

Furthermore, having the source code available (for both the model and the data analysis) also means that errors can be identified. Given the complexity of modelling work, it

should not be expected that computational modelling research is immune to the sort of software bugs which plague all programmers. This is not a problem unique to computational modelling; indeed, all sciences have the potential for undetected errors in analysis or methodology. But, the availability of source code can make computational modelling into a domain where *these sorts of errors can be found and fixed*. This is a unique opportunity for a scientific field, and as such computational modellers need to recognize that these sorts of errors will happen, and that finding them and correcting them should be a natural part of the field.

Having such source code available simplifies the task of other researchers who wish to work with multiple sorts of models. Furthermore, having access to all of the raw data (both from the modelling results and from the real-world comparisons) resolves the problem of space limitations in publications. Within the publication, the researcher can present those aspects of the data which they believe best demonstrates the capabilities of a model, and any reader interested in other aspects can do easily perform such analyses separately using the provided source code.

5.5 Communication Infrastructure

It is quite clear, given the issues raised in this chapter thus far, that having more replications and reuse in computational cognitive modelling would be of great benefit to cognitive science as a whole. However, this point has been made in various forms for as long as computational modelling has existed. The fact that it is not done more is somewhat surprising, but could be due more to practical problems than to theoretical

considerations. It may be difficult to create reusable models, or difficult to install and run models created by others. Certainly, there is currently little academic reward for doing so, and the uncertainties involved in allowing other researchers to examine the original source code (discussed in the previous section) may overwhelm the desire to advance science in this way.

These issues make it more difficult for communication about models to occur. However, some of them can be address by purely technical considerations: software tools that make the process simpler. While these tools do not represent any sort of theoretical breakthrough, they are meant to simply ease the process of creating clear, readable, and reusable models, and the storage of such models so that they can be found and made use of by other researchers.

5.5.1 Modelling Toolkits

For most of the history of computational modelling, the standard approach has been to create models from the ground up, rather than making use of software libraries that might support this task. This makes a certain amount of sense, since if there are few underlying similarities between the model being built and previous models, then there is little (if any) benefit from such reuse. However, this means that this approach to computational modelling requires extensive programming skills. Furthermore, even those modellers with sufficient programming skills seldom organize their implementation of the model in such as way as to encourage others to modify and make use of it. The emphasis is, quite reasonably, on simply making it run.

However, this state of affairs is starting to change. Software for developing particular *types* of models has become more common. PDP++ (O'Reilly & Munakata, 2000) provides comprehensive tools for making a wide variety of connectionist models. The Lisp implementation of ACT-R (Anderson & Lebiere, 1998) is used in the majority of ACT-R modelling, and includes numerous tools for examining the internal workings of the models. There is also a proliferation of tools such as BugWorks <<http://www.bugworks.org/>>, which allows for the easy creation and observation of simple embodied models such as Braitenberg vehicles (Braitenberg, 1984).

While these tools are specialized for creating particular sorts of models, the more important underlying theme is to allow a broader range of researchers to create cognitive models. By reducing the barrier to entry for modelling (i.e. by making modelling less reliant on the programming skill of the modeller), more cognitive scientists can make use of modelling as part of their research. Given this simplification, the resulting models are then also more likely to be reusable by other researchers, since they share an underlying common software library, and the details of the model are not complicated by more basic programming infrastructure.

As will be discussed more completely in chapter 6, part of this thesis involved developing a computational modelling toolkit called the Carleton Cognitive Modelling Suite (CCMSuite). The emphasis here has been to create a *generic* modelling toolkit (i.e. one that can be applied to any sort of modelling). Instead of specific software for creating ACT-R models or connectionist models, CCMSuite strives to support the common underlying processes for all modelling. Particular sub-components do allow for creation

of particular types of models (such as ACT-R models, connectionist systems, and evolutionary algorithms), and these are made in a consistent manner, allowing models to be built which combine these approaches. This is a novel approach to modelling, and has been a part of a computational cognitive modelling course suitable for non-programmers. It is also designed to be suitable for use by skilled programmers, and all of the modelling work in this thesis makes use of it.

Another important aspect of CCMSuite is that instead of just targeting the creation of models, it also supports the rest of the modelling process. This includes building models of the environment that interact with the model itself, recording of various measures from the model, and the analysis of the resulting data. The system is meant to cover the complete modelling process, including publicly sharing all code for the model and all information used in evaluating the model.

For details on CCMSuite, see section 6.6 and appendix A. Initial evaluations of its usefulness can be found in section 6.8. It is available online at <http://ccmlab.ca/ccmsuite.html>.

5.5.1 Online Repositories

While creating readable and reusable computational models without requiring extensive programming skill addresses one aspect of the communications problem, there is still the separate problem of being able to find models created by other researchers. Given the ubiquitous use of the Internet among academic researchers, it is becoming more common for individuals to make their work available for download on their academic website, or

to note that the source code is available by e-mail request. While this is a step in the right direction, this ad-hoc approach requires individual researchers to maintain these websites, adding an extra administrative burden.

One improvement on this can be seen at <http://act-r.psy.cmu.edu/models/>, where the ACT-R community has created an archive of ACT-R models. This is maintained on a volunteer basis, where individuals who have created ACT-R models that have been part of published papers can submit the original source code. While it is certainly not a comprehensive list of ACT-R modelling (less than 40 models are currently on the site), it makes these models significantly more easily accessible to other researchers.

There are also ongoing attempts to develop more generic on-line archives for computational models. The author of this thesis is involved in a project with the University of Michigan Center for the Study of Complex Systems to create an Infrastructure for Scientific Papers with Open Communication (I-SPOC). Available on-line at <http://ispoc.cscs.lsa.umich.edu/>, this site allows for users to upload any sort of computational model, including social models, environmental models, and cognitive models. These models can be accompanied by instructions for their use, and are classified according to keywords making it easier to find and use models of interest. There is also a discussion and feedback system, allowing people who have attempted to make use of the models to provide comments on their experiences doing so. Furthermore, the submitted models become non-refereed publications, giving researchers a consistent way of referring to them, and giving the creators of the models an academically recognized incentive for submitting them.

These sorts of facilities for communication of models are likely to become more common, and will encourage more transparency in the modelling process. It is hoped that, eventually, any publication which makes use of a computational model will also make that model directly and easily accessible to anyone who is interested.

5.6 Summary

5.6.1 State of the Art

- There tends to be insufficient published data to successfully replicate modelling results. This appears to be due to the inherent complexity of models, which cannot be concisely summarized in a standard publication.
- It is also difficult for modellers to make use of others' models. This has led to there being very few comparisons between models, and few examples of other researchers expanding on models originally developed elsewhere.

5.6.2 Background

- It is very important to accurately communicate the models being created. Otherwise, the scientific progress that the represent can be completely lost. For example, there are no existing working versions of SHRDLU which can perform the tasks widely associated with this important early cognitive model.

5.6.3 Contributions

- It was argued that publication of modelling source code (including the model of the environmental situation, and the data used for comparison, and any other

information needed for replicating the analysis of the model) should become part of the modelling process. This would allow the model to be replicated and expanded by the wider research community.

- Two tools to support this process were offered. The first, CCMSuite, will be used throughout this thesis to present clear and adaptable implementations of models, plus the associated tools for performing model analysis and providing the results on-line. The second, I-SPOC, provides a stable repository for such information.

6 A Methodology for Computational Cognitive Modelling

The previous chapters have described a variety of aspects relevant to producing computational models of cognitive processes. For clarity and convenience, this chapter provides a concise summarization of these points, with the intent of producing a step-by-step guide to modelling applicable to all cognitive science. This approach is agnostic to the sort of models (symbolic, connectionist, etc.) used, and to the programming language or tool-set involved. A flowchart summary is provided as Figure 6.1.

6.1 Step 1: Experimental Situations and Measures

The first stage of cognitive modelling work should involve carefully defining the aspect(s) of cognition that are to be modelled. These aspects must be observable behaviours of real cognitive systems. While qualitative measures are possible, it is preferred that these behaviours have particular quantitative measures that can be made. This can be any numerical measure desired, and any standard cognitive psychology measurement is appropriate.

If possible, a variety of *types* of measures should be used. This can include such aspects as reaction times, error rates, measurements over time showing learning, interference tasks, and so on. Multiple experimental situations are also useful. Often the data will have been previously gathered by other researchers, and if so, gaining access to the original raw data is preferred. In practice, however, the data will often only be available from the overall statistics taken from existing publications. Importantly, confidence

intervals must be available for all measurements. This means that, at minimum, the mean, standard deviation, and number of subjects/trials must be known. The preferred situation, of course, is one in which the raw individual subject observations are available, since this allows calculation of confidence intervals via bootstrapping, which does not make assumptions about the distribution of the data.

Given these experimental situations and measures, a computational model of the environment must be made. This involves mapping the stimuli and responses in the real world to suitable inputs and outputs of the model. For example, instead of a visual representation of a sentence, the computational stimuli may involve representing the individual words in some other manner to the model. Depending on the situation, information like the colour, font, size, and position of the individual words may or may not be a part of this representation.

6.2 Step 2: Creating Models

Next, various different models can be created. Since multiple sorts of models are needed for comparison purposes, this process should begin with the creation of simple, minimal models. For example, a model which merely behaves randomly can serve as an initial point of comparison, indicating how well such a simple model matches the observed behaviour.

More complex models can then be created. In many situations, these models will be the novel creation of the researcher, based on an underlying cognitive theory. Such a model may be developed entirely from scratch, or based on standard cognitive modelling

components (such as neural networks), or even make use of pre-existing complex cognitive architectures (such as ACT-R). Examples of these three possibilities are given in later chapters.

If the model is meant to follow the same algorithm or processes as the real cognitive system, then the underlying steps within the model must pay careful attention to the time course of the model. That is, any aspect of *time* which is predicted by the model should be done by the model specifying how much time various internal operations require.

Multiple operations can certainly occur in parallel, but no operation can be allowed to transfer information backwards in time within the model. This constrains the model to be a *process* model. With models of this kind, behavioural comparisons based on reaction time can establish the algorithmic equivalence of the model, giving evidence that the model follows the same underlying operations as the real system.

Variations on a model should also be investigated. One standard approach to this is to remove various components of a model. The resulting models can be compared to the original model to determine how important the removed component is. This can be seen as a version of Ockham's Razor, as there is no reason to have a component in a model if it is not vital to its performance.

In addition to original models, it is vital to also consider other models developed by other researchers which may also be appropriate for the task being considered. In the case of the RELACS model discussed previously, the repeated binary choice task is so general that almost any other cognitive model could also be applied to the situation. In order to

establish that a new model such as RELACS is the best way of understanding this aspect of cognition, it is also necessary to examine how well these other models perform.

For all of these models, there may be multiple parameters. Adjusting any of these parameters results in a new model. This results in a large space of possible models, encompassing models which are completely different, models which differ by only a parameter setting, and even models which differ via non-numerical parameters. It is this space of models which is to be considered in the research. Clearly, it is impossible to consider every such model in the space, as even for a single parameter setting there are an infinite number of subtly differing models.

Decisions must thus be made to constrain the set of models to be considered. For well-established models, or ones based on previous cognitive theory, these parameters may have fixed settings or parameter ranges. New models may warrant a more careful examination of different parameter settings, while more standard models may only be examined at a few canonical parameter settings.

6.3 Step 3: Evaluating Models

Each of the models (including the models resulting from particular parameter variations) determined in step 2 must be combined with the particular environmental situations developed in step 1, and each of the measures must be made. Since most models are stochastic, they must be run multiple times in each situation, so as to get a reasonable sample of the behaviour of the model.

In order to determine how good the models are, a comparison between the models and

reality is required. As discussed in chapter 3, both behavioural and algorithmic equivalence can be evaluated by looking at the numerical match between the measures made on the model and the measures made on the real-world subjects. A perfect model would produce measures which are statistically indistinguishable from the real thing.

Since there is no way of proving that two sets of measures come from two identical distributions, this comparison can instead be addressed by determining how different the model and reality *could be*. To do this, confidence intervals on various aspects of the measures (such as their mean value, or their variability) can be made. The maximum difference between the confidence interval for the real system and the confidence interval for the model indicates the maximum amount that the model can be reasonably believed to be in error. Better models will have a smaller maximum error. This measurement is known as *equivalence testing*, and if bootstrap confidence intervals are used then it makes no assumptions as to the underlying distributions of the data. This makes for a simple, easily interpretable measurement of how well the model and reality match. Other methods provide difficult to interpret results (e.g. a correlation between model and reality of 0.83), are extremely constrained in their conclusions (e.g. RMSD of 0.06 means that the mean result from the model differs from the mean result from the particular participants in the study by 0.06, averaged over multiple conditions), or may merely indicate that they failed to find a difference between model and reality (e.g. χ^2 and *t*-test). Instead, the equivalence measure provides the upper bound on how wrong the model may be. That is, it guarantees (with 95% confidence) that the model is accurate to within a given range. This sort of conclusion indicates the quality of the model by indicating

exactly how much it can be relied upon. None of the other measures does this.

When there are multiple measures to be considered (as there usually should be), there is no simple way of combining these numbers into a single measurement. A particular model may match well on certain measures but not on others. This can be seen as an indication of the domain of applicability of a model, as it can be used to form predictions about certain aspects of cognitive behaviour but not others.

However, if measures are of the same sort (i.e. are in the same units, such as error rate measurements in two different situations, or reaction times at two different points in time), they can be combined. In this case, taking the maximum difference (rather than the average difference) gives the more meaningful result. In other words, one might say that a model always has a reaction time within 200 milliseconds of the real system, as opposed to saying that the average difference between the model and the real-world data points is 150 milliseconds. Using the maximum difference for combining measures avoids obscuring particular measures that may be extremely poorly matched by the model.

It is also possible to relativize any set of measures so that they may all be combined. This is done by choosing a limit as to how large an error one is willing to accept for each particular measure. The equivalence measurement can be divided by this number, resulting in a value scaled so that any number below 1 indicates a model that is sufficiently close to the real situation. Once scaled in this way, the maximum value of all the measures can be found. While there is no objective means to choose such a threshold,

at the very least it should never be smaller than the confidence interval for the empirical data (i.e. the models should not be expected to be more accurate than is warranted by the available empirical data).

6.4 Step 4: Describing Models and Results

All of the source code for the models and the environments must be available for any published computational modelling research. Journals do not yet have such facilities, but the wide availability of on-line storage via the Internet makes it the most obvious means of distribution. Everything should be made available, including all of the various environments for the models (and any auxiliary information such as what exact stimuli are used) and the models themselves.

While this code should be runnable by anyone else, it is not necessarily incumbent on the creators of the initial model to produce an *easy-to-use* version of their own model.

Oftentimes the process of creation of a model is highly exploratory, and the resulting source code may not be in a simple, organized format. This should be seen as a normal state of affairs, and this thesis should not be seen as a call to force modellers to publish cleaned-up, clearly organized versions of their models for others to use. Producing such models would be useful, and could certainly help encourage other modellers to make use of the model, but should not be seen as necessary.

However, this means that re-implementing existing models can be a useful part of computational cognitive modelling research. Such re-written versions can be produced by people other than the original researcher who proposed the model. Often this will be

someone who wishes to compare the model against their own, or some other model, in situations for which the original version of the source code could not be easily adapted. Such re-implementations can also be used to ensure that the particularities of a given implementation of a model do not, in fact, affect the overall performance of the model (and if such effects are found, they constitute a deeper understanding of the model).

To communicate the results of the modelling research, the main criterion is to indicate the set of models that match the experimental data to within a certain degree. This involves specifying which models (of those investigated) do come closest (without over-fitting) to the observed data, and which ones do not. In the case of models which differ only by a parameter setting, this leads to constraints on parameter settings which can then be applied to other situations involving the same sort of model. In the case of models which differ in type, this leads to evidence as to which sort of model is best suited to the given situation.

Whenever a wide variety of models are found to be equivalent, this is an indication that more detailed information is needed. Adding new sorts of experimental data, or new variations on the experimental situation leads to new measures. These can then be used to separate models which are otherwise as good as each other. In the ideal case, exploration of the predictions formed by the models in new situations can be used to find particular experiments which have not yet been performed that could distinguish between two models. These considerations lead back to step 1 of this process.

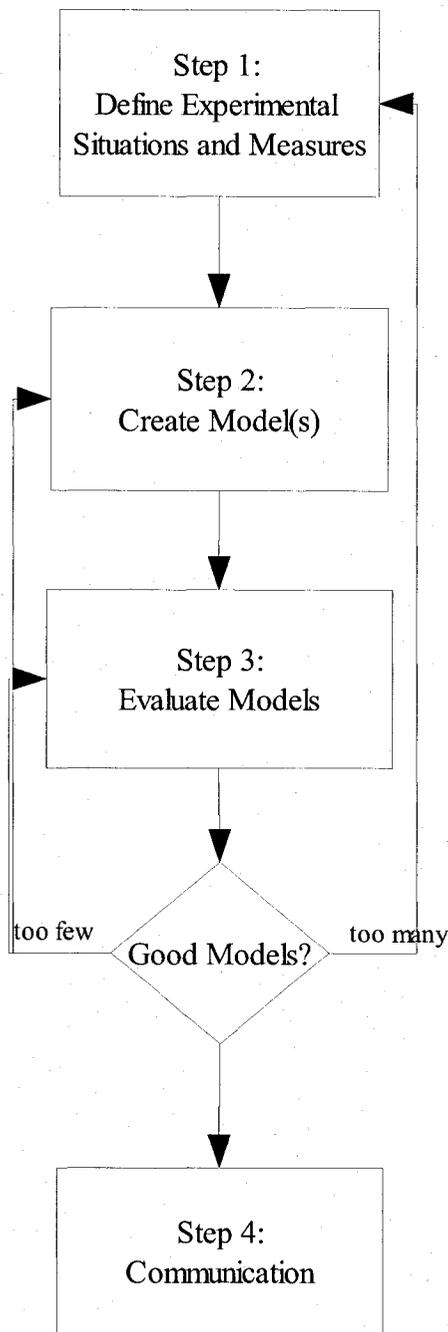
6.5 The Iterative Process of Modelling

In the flowchart below, these four steps are summarized. Importantly, there is a part of this flowchart that indicates when a modeller should go back and revisit earlier parts of this process. This happens when either too few or too many of the models are equivalent to the empirical data. There are no objective criteria for this, but certainly if *none* of the models (or *all* of them) are equivalent, then there is a need to readdress the earlier steps.

If too many (or all) of the models are equivalent to the empirical data, then this can be taken as an indication that more measures are needed. That is, the particular measures that have been chosen are too generic to give a meaningful account of the phenomenon of interest. An example of this is given in section 9.2, where a model of children's peer relations is evaluated using a measure based on a standard sociometric test. It turns out that any simple model produces equivalent results on this measure, and so it was necessary to add new measures to more adequately constrain the models. In this case, a measure of the stability over time of the sociometric measures was added.

If too few (or none) of the models are equivalent, then there are a number of possible options. The quickest approach is to remove measures from the analysis. This can result in finding models which adequately capture some aspects of the phenomenon of interest, but not all. This is demonstrated in section 9.1, where the RELACS model is shown to be equivalent to the human results in many situations, but not those where the two options give very simple rewards or very complex rewards. It is also possible to simply redo the analysis, but allowing a larger degree of error for something to be equivalent.

Another option in this situation is to examine more models. This can either be done by examining different parameter values for existing models, or by creating entirely new models. This sort of exploratory process is the core of an iterative modelling approach. One starts with a simple model, finds measures that it does not account for, and then creates new models (or modify existing models) using available cognitive theories to account for the data. By adding whole new types of measures, and by requiring the model to capture a broad range of data at the same time, it is possible to avoid the “mere parameter fitting” problem. This is especially true when a model accounts for measures of different types, and ones that were not considered when originally creating the model. Examples of this process can be seen in chapter 13 and section 14.1.



- provide a complete computational implementation of the model of the environment
- measures with confidence intervals
- definition of what is a sufficiently close match (equivalence thresholds)
- provide a complete computational implementation of the model of the real-world system
- start with a simple (usually random) model
- develop multiple types of models
- process models are preferred
- use existing architectures, if possible
- choice of parameter settings
- run repeatedly for each parameter setting
- stop when model CI \ll experimental CI
- do equivalence measurement
- if all models good, add more measures
- if no models good, reduce measures, increase thresholds, expand parameter search, or add models
- identify good parameter regions (often in separate clusters)
- if few good models, repeat with more fine-grained parameter values (near the few good models)
- all source code and data from the above
- identification of good model settings (canonical parameter values)
- description of experiments that would distinguish among the good models

Figure 6.1: The computational cognitive modelling methodology flowchart

6.6 Comparison to Other Methodologies

There are two fundamental differences between the methodology presented here and the

standard approaches used and advocated in cognitive science. Firstly, the goal of this method is to find sets or ranges of models, all of which are considered to be equivalent, rather than to find one particular best model. Secondly, this method indicates how the results of model analysis can be used to inform the overall modelling process, indicating when further empirical results are needed and when new models should be created.

For comparison, the approach advocated by Myung and Pitt (2002) involves first finding the optimal parameter settings for a model, and then using a measure of generalization to determine which one of many competing models is to be preferred. That is, for each different type of model, the parameters are varied to find the one that best fits the observations. This can be done in a variety of ways, including Mean Squared Error, although they suggest a search heuristic that maximizes the likelihood of producing the observed empirical data. Once this optimization of numerical parameters is complete, structurally different models (i.e. models with different non-numerical parameters; see section 4.3.6) are compared using a separate model selection method. This is meant to avoid over-fitting by eliminating models that match the observations extremely well, but do not generalize to new situations. This can be as simple as cross-validation (where the model is optimized using a sub-set of the available data, but evaluated using the remainder of the data) or as detailed as measures of the structural complexity of the model itself (usually in terms of the number of free parameters).

The Myung and Pitt approach results in a determination of a single best model for the available data. The goal is that this model will be the simplest one that is capable of capturing the underlying trend of the real-world behaviour, without being so complex that

it falls prey to over-fitting due to sampling noise. However, there is no mechanism for discovering when there is *insufficient evidence*. That is, the approach will always indicate that some model is the best, even when there is no statistically significant information that supports one over the other.²⁷ While this may be appropriate when finding the single most predictive model, it does not fit with the modelling goal of determining what set of models *could* provide an explanation of a given phenomenon. Any such single-result methodology has the potential for prematurely discarding models which may turn out to be the correct explanation (once more empirical results have been gathered). Avoiding this problem is a major benefit of the methodology presented in this thesis.

The other major benefit is the close connection between analysis results and the overall modelling process. Interestingly, there is a variation of the Myung and Pitt methodology which makes a somewhat similar, although more limited, connection. The Generalization Test Methodology (Busemeyer & Wang, 2000) uses *different measurements* for optimizing the model and for evaluating its fit. That is, the model's parameters are chosen based on how well it fits to one set of measures, and then it is evaluated on a completely different set of measures. Choosing these different measures is a difficult, subjective problem, but this approach can be seen as comparable to the suggestion given in this thesis that if there is a large space of models that are good, this is an indication that more measures should be added so as to distinguish between these models.

²⁷ They do note that the Model Selection Test (Golden, 2000) could be used to determine when there is insufficient evidence to choose one model over another. However, this does not generalize well to comparing more than two models at a time.

Another methodology worth noting is that presented by Roberts and Pashler (2000).

They advocate determining what a model predicts by looking at the behaviour of the full range of parameter settings available. That is, if a model does not specify a particular parameter setting, then the behaviour of the model should be thought of as a combination of all the possible behaviours from all the possible parameter settings. This differs from the methodology presented here, where each parameter setting is thought of as a separate model. This range of possible behaviours is then combined with the confidence interval range observed in the real-world data.²⁸ If the observed real-world measurement is within this range, then the model is considered to be consistent with this observation. However, this evidence is only considered to support the model if there are plausible results that are *outside of this range*. That is, positive evidence for the model only occurs when the observation is consistent with this model but inconsistent with some other plausible model.

The advantage of this approach over that of Myung and Pitt is that there is an indication as to when there is insufficient evidence for the model. If the model is highly unspecified (i.e. has a wide parameter range), or if the empirical results are not tightly constrained (i.e. have a wide confidence interval), then there will not be strong support for the model. Having this wide range means that any plausible result will be within that range.

Although Roberts and Pashler do not explicitly state this, it is reasonable to use such a negative result as an indication that new models or more empirical results are needed.

While this does provide a link between modelling analysis and the overall methodology,

²⁸ The confidence intervals of the model's data are not considered by Roberts and Pashler (2000), but could reasonably be added as well.

this approach has two major drawbacks. First, it requires a subjective determination of what results and models would be *plausible*. Second, it combines together all possible parameter settings for a model into one measurement. There is certainly room for specifying constraints on these parameters in the methodology, but it does not allow for more complex interactions between parameters. For example, as will be shown in Chapters 8 and 9, good models can occur in separate areas of the parameter space, and these regions of good models can have complex shapes. This does not fit well into the Roberts and Pashler approach.

6.7 Carleton Cognitive Modelling Suite

The methodology for computational cognitive modelling described above involves a number of techniques that are not standard practice. In particular, the creation of models with an underlying time course is not straightforward using current programming techniques, and the statistical analysis required by the equivalence testing process is not one that is found in common analysis toolkits such as SPSS.

With this in mind, one important aspect of this thesis was the creation of a suite of modelling tools which can support this process. This system is known as the *Carleton Cognitive Modelling Suite* (CCMSuite), and is a freely available toolkit for creating models, running multiple simulations, analyzing the results, and publishing those results in an on-line form. This toolkit is used for all of the models discussed in this thesis, and is meant to be suitable for any computational cognitive modelling endeavour. Complete technical documentation for the toolkit can be found in Appendix A.

It should be stressed that CCMSuite is only one way of following the methodology described in this thesis, and this thesis is about the methodology, not this particular tool. However, CCMSuite has made possible the research described in the second half of this thesis which validates the methodology. That is, by using the methodology in a variety of situations, new conclusions were arrived at which are more scientifically useful and rigorous than would have been uncovered under standard methods. CCMSuite has been an important part of enabling that process, and so warrants some discussion.

6.7.1 CCMSuite Step 1: Situations and Measures

To create model environments, CCMSuite allows for easy specification of experimental situations. In particular, it supplements normal programming approaches with the ability to easily indicate events which occur in response to other events (for example, a light turning on when a button is pressed), and to easily specify temporal delays in the environment (for example, turning a light off after two seconds). This is done in such a way as to allow multiple processes to occur in parallel.

To demonstrate this process, the following environment model is of a situation where 10 trials are repeated where the participant is allowed to press one of two buttons. A light is turned on when they are to press a button, and it turns off for 2 seconds after the button is pressed. If no button is pressed for 10 seconds, the light turns off anyway.

<pre> class ExampleEnvironment1(ccm.Model): def start(self): for i in range(10): self.light=True yield self.pressA,self.pressB,10 self.light=False yield 2 self.stop() def pressA(self): self.light=False def pressB(self): self.light=False </pre>	<pre> Repeat 10 times turn on a light wait until pressA or pressB occurs, up to a max of 10s turn off the light wait for 2 seconds (with the light off) stop the simulation if pressA occurs, turn off the light if pressB occurs, turn off the light </pre>
--	--

Figure 6.2: Complete source code for an example environment demonstrating the basic timing and triggering capabilities of CCMSuite.

It should be noted that a reader with no programming background is not expected to completely understand this example. However, it can be noted that the intent behind this tool for defining environments is to make the model itself as simple and short as possible, so as to not obscure any important details. It should be possible to directly translate the code in the model into an understandable description of the situation, as shown in Figure 6.2.

In order to record the chosen measurements from the model, a method is needed to log particular values from the model, so they can be examined after the simulation has been run. This is done via the CCMSuite logging capability, demonstrated in the following example. In this case, the measure is the Pmax measure used in the RELACS repeated binary choice experiments. This is the percentage of time that a particular one of the buttons is pressed. This is determined by counting the number of times A is pressed, and then dividing by the total number of trials (in this case, 10).

<pre> class ExampleEnvironment2(ccm.Model): def start(self): self.countA=0 for i in range(10): self.light=True yield self.pressA,self.pressB,10 self.light=False yield 2 log.Pmax=self.countA/10.0 self.stop() def pressA(self): if self.light is True: self.countA+=1 self.light=False def pressB(self): self.light=False </pre>	<p>Start the count of the number of times A is pressed at 0</p> <p>Divide the number of times A is pressed by 10, and record that in the log as Pmax.</p> <p>If pressA occurs, and if the light is currently on (as it must ignore button presses when the light is off), then increase the count of A presses by 1</p>
---	---

Figure 6.3: Complete source code for an example environment demonstrating the measurement and logging capabilities of CCMSuite.

This process can be followed to log as many different measures as desired. All data from these measures will automatically be recorded when the simulation is run, so as to allow statistics to be gathered and analyzed.

6.7.2 CCMSuite Step 2: Models

Creating models using CCMSuite is identical to creating model environments. Indeed, both the model of the environment and the model of the real system can be thought of as models in their own right. However, all of the methods for constructing models apply to both the environment around the system being modelled and the system itself. Unifying these two issues means that separate techniques are not required for defining environments and making models. For example, the timing facilities demonstrated in the previous section for environments are also used to organize the temporal structure of the model. This forces any timing measure of the model (such as reaction times) to be built up from timing information from the individual operations which make up the algorithm

of the model. This in turn allows the timing information to be used to establish the algorithmic equivalence of the model and the real system (as discussed in chapter 3).

As an example, the following simple model is meant to interact with the environment defined in Figure 6.3, above. It simply waits for the light to turn on, and then presses A. This process is repeated forever (or until the simulation stops).

<pre>class Modell(ccm.Model): def start(self): env=self.parent while True: while env.light is False: yield 0.01 env.pressA()</pre>	<p>The parent of a model is its environment; refer to it as env</p> <p>Repeat the following forever:</p> <p>While the light is not on, Wait for 10 milliseconds Now press A</p>
--	---

Figure 6.4: Complete source code for an example model demonstrating the basics of model creation and interaction with the environment in CCMSuite.

CCMSuite allows for the creation of arbitrarily complicated models. To assist this process, it comes with a large collection of standard components and operations that are commonly seen in computational cognitive models. This includes all of the following:

- Genetic Algorithms (Holland, 1975), Genetic Programming (Koza, 1992), and Evolutionary Strategies (Rechenberg, 1994)
- Multi-layer Perceptrons, including back-propagation of error learning (Rumelhart, Hinton, & Williams, 1986)
- Simple Recurrent Networks (Elman, 1990)
- Kohonen Self-Organizing Feature Maps (Kohonen, 1995)
- Adaptive Resonance Theory (Grossberg, 1987), including ARTMAP, Fuzzy ART, and Fuzzy ARTMAP (Carpenter et al, 1992).
- Reinforcement Learning, including TD-Learning, Q-Learning and SARSA (Sutton & Barto, 1998)

- ACT-R (Anderson & Lebiere, 1998)

It also includes a special facility for creating grid-based Cellular Automata environments, allowing models to be embedded within a virtual body in such a world. A model can be built using any combination of these components, and each of the components is implemented in such a way as to simplify this use of the models. These components have formed the basis of a computational cognitive modelling course which allowed non-programmers to create and build models using these systems (Stewart, 2004).

Of these modelling components, only one has been developed to such a degree to have specific suggestions for the temporal aspects of models created using it. This is the ACT-R cognitive modelling architecture, which is discussed in more detail in chapter 12. The version of it that is a part of CCMSuite offers various different procedural and declarative memory systems which can be used as part of any model, and integrated with any of the other modelling components.

By providing this broad range of modelling components, CCMSuite allows models that make use of such standard systems to be implemented quickly and simply. This makes the process of comparing multiple models significantly more feasible.

6.7.3 CCMSuite Step 3: Evaluation

The first stage of evaluating the models is to execute them multiple times, gathering the resulting measures. This usually must be done for a large number of different parameter settings, requiring thousands of separate runs. To facilitate this process, CCMSuite provides a mechanism for automatically farming out this process to any number of

separate computers and collecting the results. This involves running a client program on each of the computers, and a central server on the computer that is to collect the results. The models to be run are transferred to the clients via the Internet, executed, and then the logged measures (as described in step 1, above) are reported back to the server. These are then stored in a database for further analysis. Software running on the server allows for the specification of parameter values to investigate.

Once sufficient simulations have been run,²⁹ CCMSuite also supports the relativized equivalence measure discussed in section 4.2.8. Confidence intervals for all measures are calculated using the bootstrap approach (or, optionally, using the t and χ^2 rules if the distribution is assumed to be normal) with a configurable confidence (usually 95%). Sets of measures can then be selected, and combined by taking the maximum error. This value is then able to be compared to the empirical data, which can be given to the system either in summary form (means and standard deviations) or as raw data. The only statistics currently available are the mean, standard deviation, skew, and kurtosis.

To compare models of varying parameter settings, CCMSuite directly generates, via a point-and-click interface, a variety of graphs. This includes simple plots showing the effects of adjusting a single parameter as well as contour plots showing the effects of two parameters changing at once. This allows for the quick identification of what parameter settings result in good models.

²⁹ CCMSuite has no built-in stopping criterion for when sufficient simulations have been run. It would be possible to add such a condition, such as once the confidence intervals for the model data are much smaller than the confidence intervals for the human data.

6.7.4 CCMSuite Step 4: Communication

One important feature of CCMSuite is that all of the software is controlled via a web-based interface. That is, the exact software used to run and control the model evaluation process is able to be accessed over the Internet. When a modelling researcher is ready to publish results, the complete set of raw data, and all of the software for the models and situations can be made available to other researchers. This is done by running a server program on a machine that is accessible to the wider Internet. This information is also easily organized into a downloadable package. All of the software (for the models and the analysis) is runnable on any modern operating system.

To further assist in the communications process, the graphs generated by CCMSuite can be set to be of publication quality. All of the plots given in this thesis were created from within CCMSuite, and have a highly configurable appearance. The underlying software uses the matplotlib library, available at <http://matplotlib.sourceforge.net/>, to generate these images.

6.8 Applying and Validating the Methodology

The methodology for computational cognitive modelling that has been described in this chapter is developed from the theoretical considerations of the previous chapters. What still must be done is to investigate the practical application of this method. This occurs throughout the remainder of this thesis. The purpose of this practical investigation is two-fold. First, these examples validate that this methodology can be used in these situations to produce important cognitive modelling results. This process provides details of how the methodology is appropriate to various situations. Second, each of the

situations to be discussed is the subject of ongoing cognitive science research. By applying this methodology to these situations, the result should be a more correct and scientifically useful portrayal of the current state of affairs in the quest to understand these aspects of cognition.

The examples used in chapters 7 to 13 are from three different cognitive research areas. The first of these is the repeated binary choice task that was introduced for the RELACS model in section 2.3. Here, participants are repeatedly asked to press one of two buttons, and feedback is given after each button press. This is a standard cognitive psychology experimental task that has been well-studied (Siegel & Goldstein, 1959; Myers et al, 1963; Friedman et al, 1964) in terms of human performance in various situations.

The second domain is from social psychology, and involves the development of popularity, rejection, and neglect in groups of school children. Here, a simple test requiring individuals in a group to identify three people they like and three they dislike results in an overall measure categorizing individuals into one of five categories: Popular, Rejected, Neglected, Controversial, and Average. Since this measure is the most common one used to identify children at risk of developing social problems, a model of the process involved may lead to a deeper understanding of the situation.

The final domain to be considered moves away from modelling human behaviour and is instead a foraging task similar to that seen in ants. By keeping track of particular aspects of the environment, creatures are able to efficiently seek out locations in the environment (such as food sources) and then return to their homes. This situation is a departure from

most of those considered in this thesis, as there is no direct comparison being made to a particular real-world system. It is included to demonstrate how the methodology can be applied in these exploratory situations.

To allow complete communication about these models, and to provide direct access to the raw data and analysis tools, everything about these models can be found the following website: <http://ccmlab.ca/ccmsuite.html>. From there, each of the models discussed can be examined along with their various environments. The software can also be downloaded, allowing further simulations to be run to investigate scenarios not considered here. This system is all part of the Carleton Cognitive Modelling Suite and is meant to supplement the discussion given in the following chapters, as per the considerations of communication discussed in chapter 5.

6.9 Applying and Validating the CCMSuite Toolkit

Although the primary purpose of this thesis is to develop a methodology, it has become clear that a software toolkit developed specifically for the methodology could greatly improve the ease with which the methodology is followed. This would, in turn, encourage greater use of this methodology in the wider cognitive science community. For this reason, the CCMSuite was designed to be used by any cognitive scientist, and to not require an extensive computational background. It is meant to guide scientists through the process of modelling, and to create clear, understandable, and sharable models.

However, evaluating this system to determine whether it is successful at these tasks is a

difficult process. A standard problem in educational research is that the expectations of the learners greatly affects their overall performance, and simply introducing something new reliably improves performance (e.g. Clark & Sugrue, 1989). This effect, sometimes known as the Hawthorne effect or, perhaps more accurately, the novelty effect, makes it especially difficult for a researcher who has developed a particular tool to be directly involved in its effectiveness. It is always possible that any increase in performance of the people using the tool is merely due to the enthusiasm of the person teaching the tool, or merely the fact that the tool is something new. For this reason, it was decided that a rigorous, controlled study of the usefulness of the CCMSuite toolkit itself is outside the scope of this thesis. Demonstrating that the toolkit is useful is not necessary for concluding that the overall methodology is useful.

However, there have been a variety of anecdotal responses to CCMSuite that are worth reporting. Overall, they indicate that CCMSuite can be successfully used by non-programmers to create useful models as part of cognitive science research. As reported elsewhere (Stewart, 2004), CCMSuite has been used as part of a graduate level computational modelling course (CGSC 5001) that is required for the Cognitive Science program at Carleton University. Students without a strong programming background (at most one undergraduate level programming course) were able to use the system to replicate and investigate classic published cognitive science models. CCMSuite is also being used for an annual summer workshop course at Carleton (CGSC 6501), where graduate level cognitive science students use it to develop models relevant to their own research interests.

The presence of this tool corresponds to a dramatic increase in the amount of modelling work occurring in the Carleton cognitive science program. These models are starting to appear in publications (e.g. Pyke, West, & LeFevre, 2007; Rutledge-Taylor & West, 2007), and form an important part of two other PhD theses. Moreover, students have consistently found the models created via CCMSuite to be clear and customizable to different situations. At the 12th annual ACT-R Workshop in 2005, a few attendees who had just finished an intensive week-long summer school using Lisp ACT-R commented that the ACT-R models made with CCMSuite was immediately readable and clearer to them than the Lisp ACT-R models they had been trained on. While these anecdotes are clearly not conclusive proof of the usefulness of CCMSuite, they do indicate that this approach is worth pursuing further.

6.10 Summary

6.10.1 Contributions

- A methodology was presented that can be applied to all computational cognitive modelling situations. It is based on establishing the equivalence of models (i.e., finding sets of models that are equivalent to the real system to a particular degree), rather than a best numerical fit. This reflects an emphasis on explanation rather than prediction. The method starts by creating explicit models of the environment and the measurement process (Step 1), which precisely describes what part of the real system are and are not included in the model, as discussed in chapter 2. Creating models that can involve internal temporal processes to generate time predictions (Step 2) allows for the evaluation of algorithmic

equivalence as well as behavioural equivalence, as discussed in chapter 3.

Evaluation of these models (Step 3) using the equivalence measure from chapter 4 provides a rigorous test for the quality of the model and the effects of various parameter settings. Describing these results (Step 4) in a complete and repeatable manner addresses the issues raised in chapter 5, making the model accessible for use and scrutiny by the wider scientific community.

- The CCMSuite toolkit was developed to support this process by providing tools for creating models and environmental situations, evaluating models, performing statistical analysis, and making all of the code, data, and results available to other researchers. Initial results indicate that this system is usable by a wide range of cognitive scientists, and allows them to more rigorously examine the quality of their models than do the alternative approaches discussed.

7 Basic Model Evaluation

The simplest example of applying this computational cognitive modelling methodology involves a single model in a single environment, with only one measure being made. To demonstrate this, the RELACS model introduced in section 2.3 can be used, performing the simple repeated binary choice task for which it was developed by Erev and Barron (2005).

7.1 Environment

In the paper describing the RELACS model, a variety of environments and measures are used. For the purposes of this chapter, only a single measure in a single environment will be considered. In this case, the environment will be the one identified as Experiment 2 in the paper. Here, there are 200 repetitions of pressing one of two buttons. One button always gives a reward of 11, while the other button gives a reward of 19 half of the time, and 1 the other half of the time. The measure to be made is the proportion of the time the first button is pressed in the last 100 repetitions. This is referred to as P_{max2} , as it is the probability of choosing the option with the maximum expected reward in the 2nd block of 100 trials. A summary of the situation can be seen in Figure 7.1.

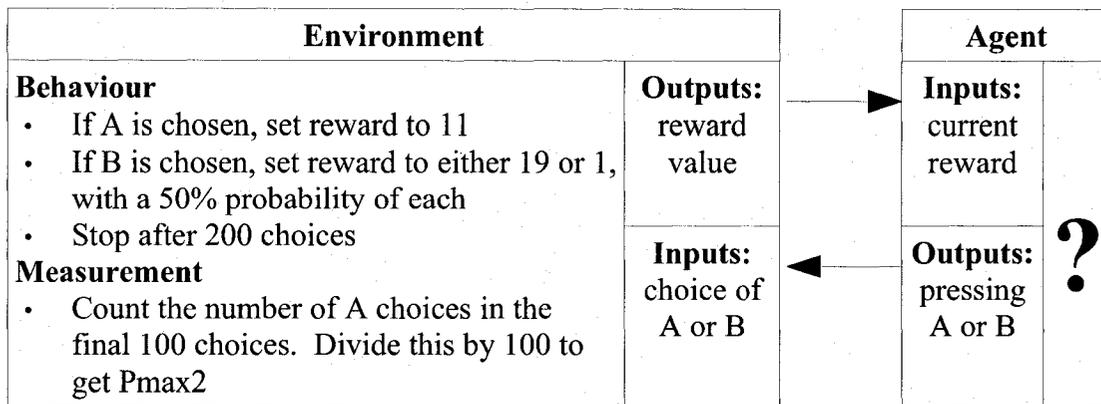


Figure 7.1: Agent-Environment interaction and environment specification for experiment 2 (Erev & Barron, 2005).

In order to map the actual real-world experimental situation into this model of the environment, a number of simplifications are being made. As discussed in section 2.2, these simplifications are in three domains: the materials/stimuli, the instructions, and the actions. For the stimuli presented, the display of the reward value is done as a simple numerical input to the model, as opposed to a visual display. This bypasses the problem of visual recognition, and assumes that there is some process which reliably interprets the numerical value on the screen seen by the participant, and that this value is available for further processing. This process is not part of the model. For the instructions, it is assumed the participant stays on task and does not choose some other option (such as getting up and leaving, or falling asleep) than pressing a button. No other cognitive processes are running at the same time as this model. For the actions available to the participant, all of the complexities of motor planning and coordination are ignored. Instead, the model simply outputs a value that indicates which button is to be pressed.

It should be noted that this environmental model has no role for the temporal aspects of the situation. Time is, in this case, being completely ignored, as there is no reaction time

data to compare to. The model may respond instantly, or it may press a button once every five years; the result will be the same. Since simulated time and real time (the time required for the computer to run the model) are separate and distinct, it is certainly possible for a model in this situation to indicate that all 200 choices of which button to press happen in an instant. In the CCMSuite modelling system, time only passes within a model when explicitly indicated, so since we are not specifying how long it takes for the reward to appear after a button is pressed, or any other action within the model, including the operations underlying the model, these activities will take no time at all. This would, of course, have to change if any of the measures for comparison involved time.

As is very common in modelling situations, the complete real-world experimental raw data needed for comparison is not available. However, Erev and Barron (2005) do indicate certain summary statistics, as shown in Table 7.1. These values can be used to approximate the confidence intervals needed to compare the model behaviour to the real-world behaviour.

Number of participants	14
Mean Pmax2 value	0.71
Standard Deviation of Pmax2	0.246

Table 7.1: Summary statistics for Experiment 2 (Erev and Barron, 2005)

In order to perform comparisons using these values, confidence intervals are needed. Since the complete raw data values are unavailable, it is necessary to make some assumptions about the distribution of this data. The most common assumption is that these values are actually normally distributed. Given this assumption, the confidence interval for the mean of this distribution can be calculated using the Student's t

distribution (see section 4.2.5 for details), and the confidence interval for the standard deviation can be found in a similar manner using the χ^2 distribution. This is done automatically by the CCMSuite system, resulting in the following intervals.

95% Confidence Interval of Pmax2 mean	0.568 – 0.852
95% Confidence Interval of Pmax2 standard deviation	0.178 – 0.396

Table 7.2: Summary statistics for Experiment 2 (Erev and Barron, 2005)

As can be seen in Table 7.2, even though the particular sample of 14 participants gave a mean of 0.71, the overall mean of the full population of individuals is only known to be somewhere between 0.568 and 0.852. It is this population mean that the model should be compared to, not the mean of the 14 people who happened to be used in the study. A similar reasoning applies to the standard deviation. To evaluate a model, the distribution of its behaviour must be compared to this range of values, not the particular sample data given in the original publication.

7.2 Model

The model considered here is the complete RELACS (Reinforcement Learning Among Cognitive Strategies) model, as described previously in section 2.3. Since parameter variation is not being considered yet, the model will have its parameters locked at the values specified by Erev and Barron (2005), and shown in Table 7.3.

L	8
a	0.00125
B	0.2
K	4

Table 7.3: Parameter settings of the best version of the RELACS model (Erev & Barron, 2005)

To connect the RELACS model to the environment, a mapping is needed between the outputs of the model and the actions in the environment, as well as the stimuli in the environment and the inputs to the model. In this case, the reward value must be interpreted as the feedback for the model, and the output of the RELACS algorithm must cause either button A or B to be pressed.

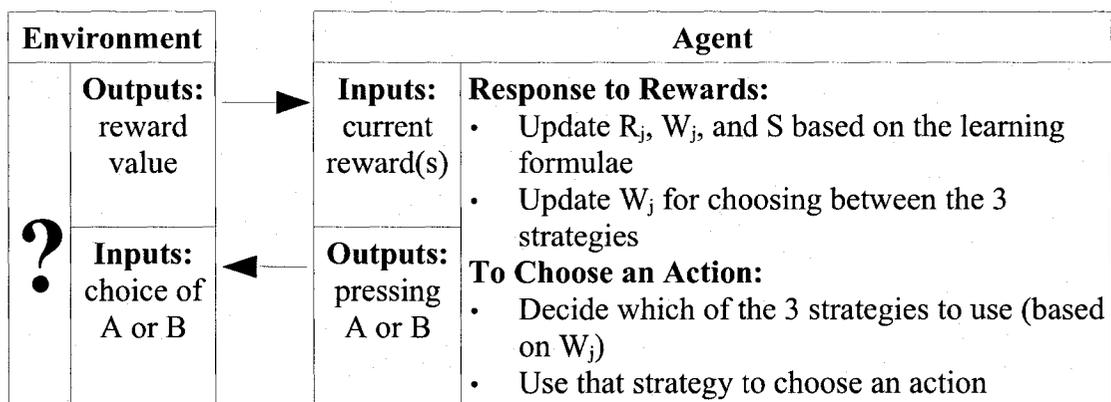


Figure 7.2: Overview of the RELACS model of repeated binary choice. See section 2.3 for more details.

A general overview of the RELACS model is shown in Figure 7.2. The inputs to this model are specific numerical values (for the rewards), and the outputs are which button is to be pressed. This means that the model does not include cognitive aspects such as the visual interpretation of the reward values, or the motor control for pressing the buttons.

The complete source code for this model can be found in Appendix B.

7.3 Evaluation

To evaluate the behaviour of the model, the same measure must be made on it as was made on the human participants. In this case, the Pmax2 value must be measured. Each time the model is run, a different Pmax2 value will be generated, as the RELACS model has some built-in stochastic behaviour. Indeed, since human behaviour always contains some degree of variation, any model of that cognitive activity should also vary by a similar amount. This is why not only the mean value of the measure will be compared, but also other statistics such as the standard deviation.

As the model is run multiple times, a set of Pmax2 values are produced. To compare this distribution to that observed in the human participants, the confidence intervals are calculated. These confidence intervals are of the mean of the Pmax2 measure, its standard deviation, and any other desired statistic. In this case, the skew and kurtosis are also measured. A skew above zero indicates the distribution has a longer tail above the mean than would be expected of a normal distribution. A kurtosis above zero means that the distribution has a sharper peak near the mean than would be expected in a normal distribution. In this case, since the human data points for comparison are unavailable and assumed to be normally distributed, this implies a skew and kurtosis of exactly zero for the human data.

The confidence intervals are calculated using the bootstrap method described in section 4.2.6, resulting in the values shown in Table 7.4.

Pmax2 mean	0.605 – 0.635
Pmax2 standard deviation	0.094 – 0.116
Pmax2 skew	-0.226 – 0.231
Pmax2 kurtosis	-0.722 – 0.080

Table 7.4: 95% Confidence Intervals for the RELACS model

Because no information is available on the skew and kurtosis of the real human measurements of Pmax2, these intervals cannot be used to evaluate the model. It may be tempting to note that both confidence intervals include the value zero, which is consistent with the assumption a normal distribution for the real data. However, this observation does not allow for any strong conclusions as to the quality of the model. The reason for this is that the confidence intervals can be reduced simply by running the model more times, and gathering more data. This means that one can *always* run the simulation enough times so that the resulting confidence interval *does not include zero*. The true value of the skew and kurtosis (or, indeed, any other statistic) will *never* be exactly zero (or at least this would be an exceedingly rare occurrence). Thus, if one was to interpret the fact that the above intervals do, in fact, include zero as a good sign for the model, this judgment would be more to do with the number of simulation runs made than anything about the model itself.

Instead, the focus must be on the two statistics that can be compared: the mean and the standard deviation of the Pmax2 measure. These values are graphed in Figure 7.3, below. The two bars on the left show the mean Pmax2 value for the model and the human participants, respectively, while the two on the right show the standard deviation.

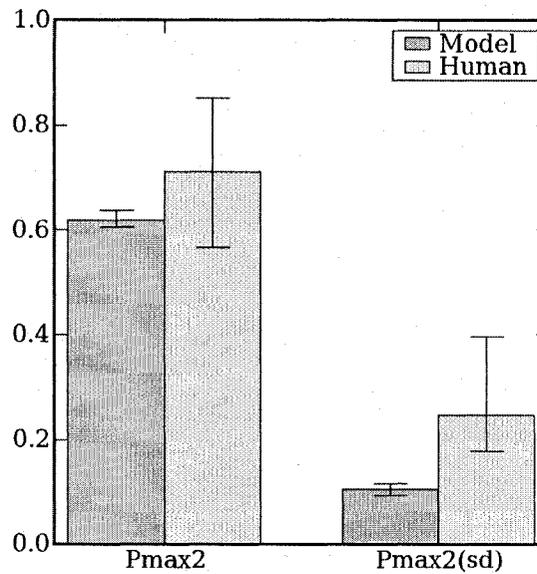


Figure 7.3: Comparison of RELACS model to human data from Experiment 2 in (Erev & Barron, 2005)

One observation that is directly obvious from this graph is that the confidence intervals for the model are much smaller than those for the human participants. This is generally the case for all computational cognitive models, as it is usually much easier to run a computer model than it is to run a human (or animal) participant through an experiment. This means that there are generally more data points available on the distribution of the measurements on the model than on the real-world system, resulting in smaller confidence intervals. In other words, it is much easier to characterize the overall performance of the model than the overall performance of the system being modelled.

This state of affairs means that a common interpretation short-cut in psychology cannot be used here. In most experimental psychology studies, the numbers of subjects in two groups that are being compared are roughly equal. This means that the size of the confidence intervals will be proportional to the standard deviation of the performance of each group. This means that some people, looking at the two bars on the left in Figure

7.3 will believe that the human performance is much more variable (has a higher standard deviation) than the model. This line of reasoning is only valid if there are the same number of subjects as there are runs of the model, which will seldom be the case.

Instead, to judge how well the model captures the variability of the human data, the two bars on the right of Figure 7.3 must be examined. These show the standard deviations of the model and the human participants. From these values it can be easily seen that the model does, in fact, exhibit less variability than is seen in the real world.

As discussed in section 4.2, the most accurate way of characterizing the above data is to determine the equivalence test threshold: the maximum difference between the confidence intervals. This value is how similar the model and real-world system can be judged to be (with 95% confidence). This is because the only thing we can be confident of is that the actual mean and standard deviation of the model and reality are *somewhere within the confidence interval*. The actual sample statistics (the height of the bars themselves, ignoring the confidence intervals) only indicate information about the particular subjects who happened to be chosen for the study, and the particular random chances that occurred while running the model however many times it was run. In other words, the fact that the human subjects happened to have a mean of exactly 0.71 is *completely irrelevant* to interpreting the quality of the model, except inasmuch as this value is used in the calculation of the confidence interval. A model which gives a mean performance of 0.71 must *not* be considered to be any better than a model which gives a mean performance of 0.85, or 0.57, or any other number within that confidence interval range. Failure to follow this advice results in “good” models which are actually only

good at matching to the particular combination of subjects who participated in the study, rather than the population at large.

For the mean, the equivalence test threshold is found by taking the lowest value in the interval for the model, and the highest value in the interval for the real-world data. This represents the worst-case scenario. Since the actual values for these two means could be anywhere in these intervals, it could be that the actual model mean is 0.605 and the actual human mean is 0.852. This results in the model having a mean which is incorrect by 0.247 (i.e. the model's probability of choosing the best option in the second block of 100 trials differs from the human subjects by 0.247).

It may be that the model is better than this. Indeed, since the confidence intervals do overlap, it is *possible* that the model exactly matches the human data (or comes very close; perhaps differing by less than 0.001). However, given the evidence available, the only conclusion that can be made is that the model does not differ by *more* than 0.247 (with 95% confidence). This is the main conclusion that can be drawn from this data.

This is in contrast to the common interpretation of data that ignores confidence intervals. Any modelling work that uses correlation or Mean Squared Difference (see sections 4.2.1 and 4.2.2) would take the difference between the sample mean of the human participants (0.71) and the mean of the model data (0.619) and state that the model differs from the real world by 0.091. This is not the case. It is true that the average Pmax2 value of the particular 200 runs made of this model differs from the average Pmax2 value of the 14 participants in the study by 0.091. However, the point of the model is to apply to this

cognitive phenomenon *in general*, not the particular results of those 14 individuals on those particular trials.

Turning to the variability of behaviour, the standard deviation of the model is between 0.094 and 0.116 (Table 7.4) and the standard deviation for the human data is between 0.178 and 0.396 (Table 7.2). This gives an equivalence test threshold of 0.302 (calculated by taking 0.396 and subtracting 0.094). It can thus be stated that the model's standard deviation differs from the real standard deviation by no more than 0.302 (with 95% confidence). In this case, since the interval for the model does not overlap with the interval for the real data, it can be concluded that the model has a lower standard deviation than the real data.

When interpreting this standard deviation information, it is also important to avoid reading too much into the fact that the confidence intervals do not overlap. It is true that, since the confidence intervals do not overlap, there is less than a 5% chance of the observed behaviour happening if the model and the real system had the same standard deviation. However, as was discussed in section 4.2.4 on the *t*-test, this is *not* useful information. *Any* model, no matter how good, can produce confidence intervals which do not overlap with the system they are modelling. This is a function of the number of trials. In this situation, with enough human participants it would be possible to show that the Pmax2 means also do not overlap. However, this would not affect the conclusion that the model is accurate to within 0.247 for the mean of Pmax2. Having more human data may allow this limit to be *reduced*, but it should not otherwise affect our interpretation of the quality of the model. Having non-overlapping confidence intervals should only be

interpreted as an indication of which *direction* the model is inaccurate in (too low or too high). It is known from the beginning that the model will not be perfectly accurate, so noting that the intervals do not overlap does not somehow invalidate the model. What modellers need to establish is what the maximum *size* of the difference between the two is, and that is what the equivalence test threshold indicates.

The result of this analysis is that the RELACS model (with this particular set of parameters) produces behaviour which has a mean Pmax2 value within 0.247 of the real-world behaviour, and a standard deviation which is at worst 0.302 below the real-world behaviour. Any use of this model for explaining or predicting human behaviour can only be considered to be accurate to this level.

By itself, these values are not particularly meaningful. To gain a more developed picture of how this model compares to others, these other models must also be analyzed. In the next chapter, this process will begin by examining the effects of adjusting particular parameters within the model.

7.4 Summary

7.4.1 Methodology

This chapter depicts the simplest possible use of the cognitive modelling methodology described in this paper. One situation was defined, with a single measure. One model was created, and all parameters were fixed. Evaluation consisted of using the equivalence test to determine how different this model could be from the real data. The result is that the model gives a Pmax2 value that is within 0.247 of the real world data.

This is relatively good, since the human results have a confidence interval of 0.568 to 0.852, giving a total range of 0.284. Given this amount of ambiguity in the human data, there is no sense in finding models that are more accurate. This means that the RELACS model with this parameter setting can be said to be equivalent to the human performance. Even though it may be in error by up to 0.247, the empirical value is not known to greater accuracy than this, so there is no sense in requiring models to be more accurate.

However, to follow the methodology, more needs to occur. Since all of the models are equivalent, the main thing that needs to occur is for more measures to be investigated. This is done in chapter 9. Since there are parameters in the model, they should also be adjusted to see their effects. This is done in chapter 8. Particular components of the model should be removed, to ensure they are required. This is done in chapter 10. Finally, alternate models using alternate architectures should be examined. This is done in chapter 12.

7.4.2 Contributions

- Using the new methodology, it was shown that RELACS with the parameter values identified in Erev and Barron's original work (Erev & Barron, 2005) produces behaviour in condition #2 that is within 0.247 of the real human data. This conclusion is 95% confident, meaning that if this process is repeated, it will be correct at least 95% of the time, taking into account sampling of both the human behaviour and the model behaviour.

8 Parameter Variation

In the previous chapter, only one model was examined. This was the RELACS model, fixed with a particular set of parameter settings. However, there are four separate parameters in the RELACS model. Adjusting any of these parameters will affect the overall behaviour. Indeed, it may be clearer to think of each possible parameter setting as a *separate model*, since without mathematical proofs, there is no a priori way to know what the effects will be of changing these parameters. Instead, these behavioural results of changing these parameters must be established by running the system with different parameter values.

The purpose of this exercise is to establish *ranges* of parameter settings which give models which accurately match the human data. The purpose is not to find one particular parameter setting which best matches the particular sampled human data. This would be a case of over-fitting the model to the data. Instead, if the equivalence test threshold is used as a measure of the difference between the model and reality, the different parameter values will provide different maximum differences. Models which differ by less than a certain amount can be considered “good” models.

To demonstrate this, the exact same scenario as used in the previous chapter will be examined. This involves only a single measure (P_{max2}) in a single experimental paradigm (the repeated binary choice task with one choice always giving a reward of 11 and the other giving a reward of 1 half of the time and 19 the other half of the time).

8.1 Changing One Parameter

To begin, only one parameter will be changed. For the first example, the effect of adjusting the a parameter will be examined. In the previous chapter, this value was set to 0.00125. This choice was based on the original paper (Erev & Barron, 2005), which indicates this is the best parameter setting found. However, the paper does not indicate what values were tried. Was 0.00124 tried? What about 0.1? 0.5? 297? Or -999999? In theory, any real value could be used in this model. On most modern computers (including the ones used for the modelling reported in this thesis), real numbers are represented by a 64-bit floating point value, resulting in 18,446,744,073,709,551,615 different possible values. Clearly, not all can be tried. Instead, a few particular values can be chosen for analysis.

The choice of parameters to examine is in some sense arbitrary and dependent on the researcher. In some situations, the structure of the model itself can limit the possible values. In some models, particular parameter values must be positive, or between 0 and 1, or integers. In this case, the parameter a appears in the RELACS formula described in section 2.3.

$$W_j \leftarrow W_j(1-a) + v_j a$$

The purpose of this formula is to take the current value for W_j and change it to be slightly more like v_j than it currently is. This can be seen as a weighted average of W_j and v_j , with the parameter a adjusting how heavily v_j is weighted. Since v_j is the most recently seen reward for making choice j , a large value for a will weight recent events more strongly.

In this situation, it would be common to limit the values of a to between 0 and 1. This is because otherwise, the formula would end up changing W_j to be *less* like v_j . This is against the *intent* behind the formula and the creation of the model, but there is no technical reason why a parameter value outside this range could not be used.

As an initial exploration, the parameter a will be allowed to vary across the range 0 to 1 and slightly outside that range. This should give a global overview of the behaviour of this parameter. All of the other three parameters will be left at the values they were at originally. The parameter values to be considered are shown in Table 8.1.

L	8
a	-0.1 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1
B	0.2
K	4

Table 8.1: Parameter settings for the RELACS model to be investigated to indicate the global effect of parameter a .

For each of these parameter settings, running the model multiple times results in values similar to those seen in the previous chapter (Figure 7.3). For now, only the mean value will be considered. These values can be graphed together, as shown in Figure 8.1. The grey background area indicates the confidence interval range of the human data for comparison.

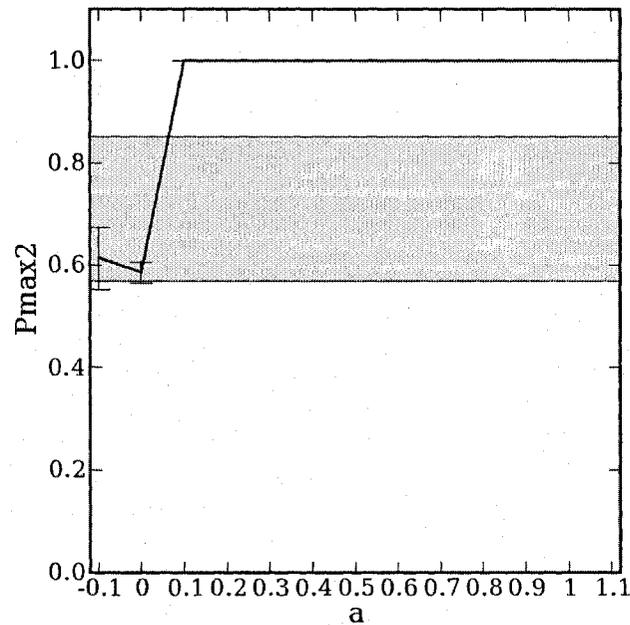


Figure 8.1: The effect on P_{max2} of adjusting parameter a over a wide range. The grey bar indicates the 95% confidence interval for the human data.

Examining Figure 8.1 reveals that for any examined value of a from 0.1 up, the model consistently gives a P_{max2} value of 1.0, which is well outside the range observed in humans (the grey region). It should be noted that this is only known for those values which were examined; it could be that the parameter value 0.672738 does a perfect job of matching the human data. However, it is infeasible to examine all possible parameter values. Instead, attention can be focused on the area of parameter values which comes closer to matching the human data. To do this, another series of simulations can be run with a different set of values for a . These values are shown in Table 8.2 and the resulting data in Figure 8.2.

L	8
a	-0.01 0 0.01 0.02 0.03 0.04 0.05
B	0.2
K	4

Table 8.2: Parameter settings for the RELACS model to be investigated to indicate the effect of parameter values a which give results similar to humans.

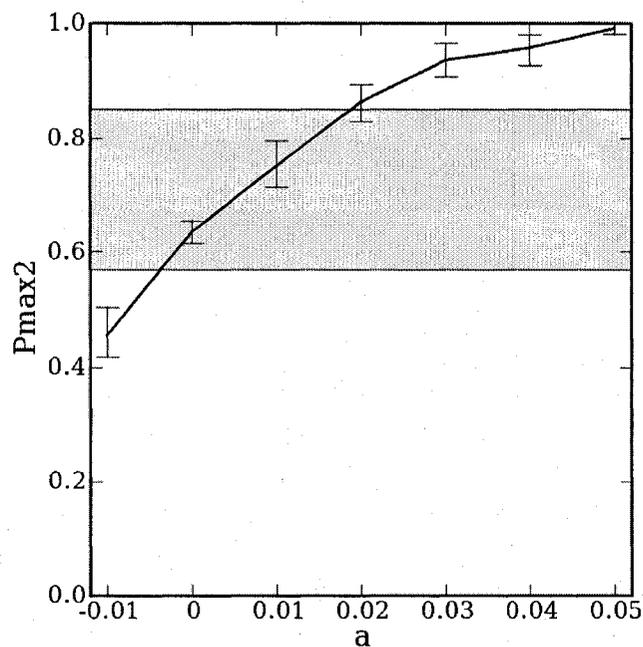


Figure 8.2: The effect on P_{max2} of adjusting parameter a over a narrow range. The grey bar indicates the 95% confidence interval for the human data.

From this graph, it can be seen that parameter values of a between just below zero to around 0.02 all lead to models which could match the real-world data. Any of the values within that range should be considered *equally good models*. This is in contrast to treating a particular value (such as 0.00125) as being somehow special. There is no evidence that any value within that range is more likely to match the real-world data than any other value.

It is also useful to plot the equivalence test threshold value directly. As discussed in

section 4.2.5, this is the maximum difference between the confidence interval for the model and the confidence interval for the real-world data. This indicates how large the difference between the model and reality could be, given the currently available evidence.

This is shown in Figure 8.3.

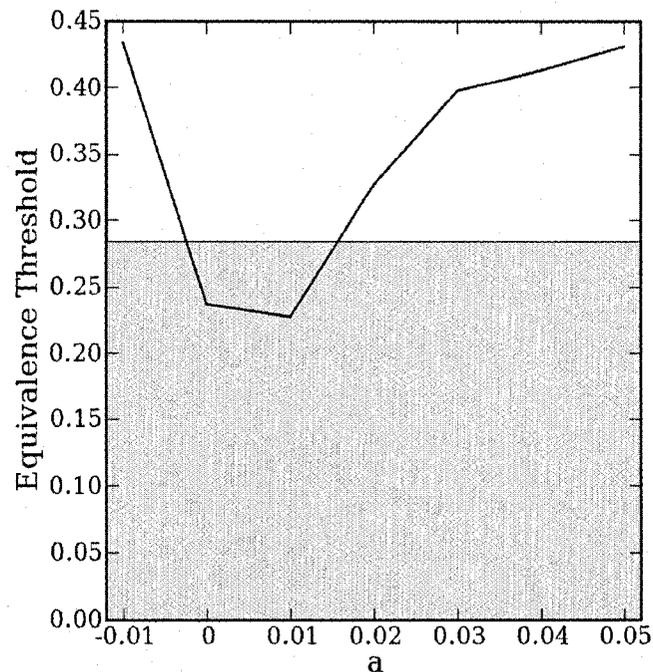


Figure 8.3: The maximum likely difference between the RELACS model and human subjects. The grey area indicates models which differ by less than the human confidence interval.

The purpose of Figure 8.3 is to clearly state what is confidently known about the model's relationship to the human data. The "equivalence threshold" indicates how different the model and reality could be. From the graph, it can be seen that an a parameter value of 0 or 0.01 both result in models that are confidently known to differ from the real human performance by less than 0.25 on the Pmax2 measure.

The grey area of Figure 8.3 indicates the size of the confidence interval of the human

data. This is included so as to indicate how good it is *possible* for a model to be. Given a confidence range of 0.284, as indicated above, it is *impossible* for a model to have an equivalence test measure of less than 0.142. This is because even a model which always gave exactly the mean result found in the human subjects could still be in error by up to half the size of the confidence interval. In other words, given the human data currently known, we cannot expect any model, no matter what, to give a result better than 0.142 on the above graph.

Also, if a model's confidence interval is entirely within the human confidence interval, then the value in the above graph will be within the grey area. This means that any parameter value which fits within the grey area is *equally good as any other model in that range*. If two models with different parameter values both give confidence intervals which are completely within the human range, then there is no reason to prefer one over the other. This is because the real human value is equally likely to be anywhere within the confidence interval. This means that once a model can be seen to be within the grey area above, it should be considered to be on par with any other model in that region. This avoids identifying one particular model as having the closest numerical fit to the particular sample data, which can be the result of overfitting.³⁰

At the same time, and seemingly paradoxically, a model with a lower equivalence test threshold can still be confidently said to be closer to the real world value. In the above graph, the model with $\alpha=0.01$ is known to be closer to the mean performance value of the

³⁰ This distinction between overfitting to sample data and a more general characterization of the data is similar to the distinction between "optimality" and "honesty" in statistical modelling (Breiman et al., 1984; Denison, 1997).

particular people in the study than the model with $a=0$. However, both models have the same chance of being correct (i.e. of having exactly the same mean as the overall human data). This distinction is important to keep in mind, as it impacts the use of models. If all that is needed is a model that produces results that are the most *predictive* of the available human data, then one can simply use the model with the lowest equivalence test threshold. If, however, one is interested in looking at *explanations* of a particular cognitive process, then all the models with low values must be considered to be equally possible.

One other thing to consider about this graph is that it is highly conservative. If the model is run more times, gathering more data, these values can and will change. However, it is highly unlikely (no more than a 5% chance) for any of the data points to move upwards. This is because gathering more data on the simulation is likely to move the average closer to the true average, which is within the confidence interval 95% of the time. This means that the only motion likely to occur in the above graph if more values are gathered is movement downwards. As an example, this means that the model with $a=0.02$ *might* have an equivalence threshold below 0.284 (the grey line), but right now there is not enough evidence to confirm this. Gathering more data can improve these results (i.e. reduce the maximum difference between model and reality), but is unlikely to make things worse.

From the data in Figure 8.3 it can be seen that parameter values for a in the 0 to 0.01 range give equally good models. It is desirable to gain a clearer understanding of what happens between 0.01 and 0.02, so another series of simulations can be run with

parameter values in this range.

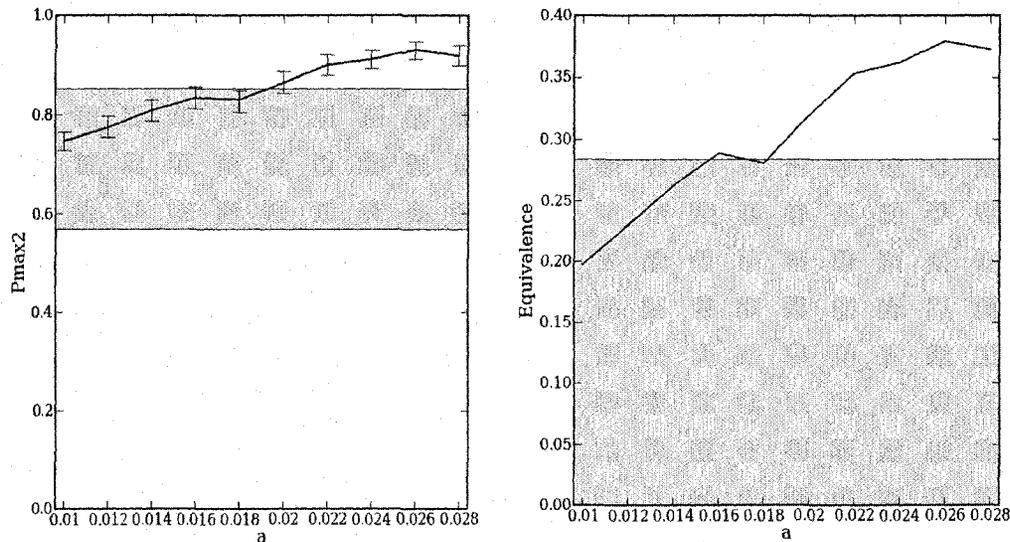


Figure 8.4: The effect on P_{max2} of adjusting parameter a over a very narrow range, and the corresponding equivalence data.

These graphs indicate that any value of a below 0.02 results in a reasonable model.

Above 0.02, the maximum difference that could occur between the model and reality increases sharply, and below this value consistently produces models which differ from reality by less than the size of the human real-world confidence interval.

The conclusion is that, if the other three parameter values are fixed at their current settings, then adjusting the a value causes the model's performance to vary. Given the uncertainty about the real-world human data (i.e. the size of its confidence interval), the best that can reasonably be expected for a model to perform is somewhere between a maximum difference (i.e. equivalence test threshold) of 0.142 to 0.284. In this case, such models are found when a is less than 0.02, down to just below zero. It is, of course, technically possible that some parameter values that are between the ones that have been

tested would also result in models that are equivalently good, but none have been found.

Each of the other parameters could also be varied. If the same analysis done above is performed on the parameters B , L , and K , then the graphs shown in Figure 8.5 are the result. For each of these graphs, the other parameters are left at their default values.

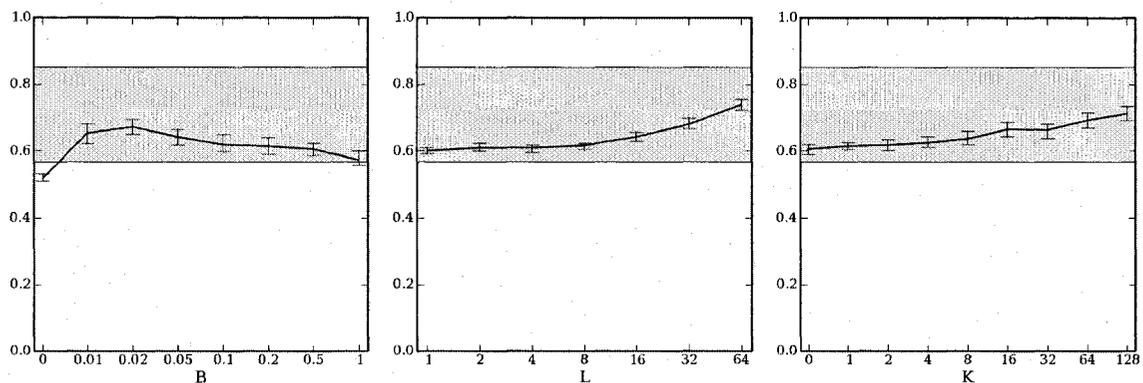


Figure 8.5: The effects of adjusting parameters B , L , and K individually, while leaving all other parameters at their default values.

These graphs show a stark contrast from the results of adjusting a . Instead of finding equivalent models for only a narrow range of values, the models produced by adjusting B , L , and K are almost always equivalent to the human data. The only exceptions are models with extremely low or extremely high values of B .

8.2 Changing Two Parameters

All of the above analysis was performed when varying only one parameter. In other words, the fact that good models are produced when a is between 0 and 0.02 may be entirely dependent on the values of the other parameters. Perhaps if B is 0.3 then good models will only appear when a is between 0.3 and 0.32, instead. In other words, there may be interaction effects between these parameters. These possibilities can be

investigated in a similar manner as the single parameter case, but with multiple parameters changing.

As a first example, the parameters a and B can both be varied. These parameters were chosen because they provided the largest behavioural effect, as shown in the previous chapter. The values to be tried are shown in Table 8.3, below. Since there are six values of a considered and eight values of B considered, this results in 48 different models which must be run. The results from these runs can be seen in Figure 8.6.

L	8
a	0 0.01 0.02 0.03 0.04 0.05
B	0 0.01 0.02 0.05 0.1 0.2 0.5 1
	1
K	4

Table 8.3: Parameter settings for the RELACS model with two parameters being varied at the same time.

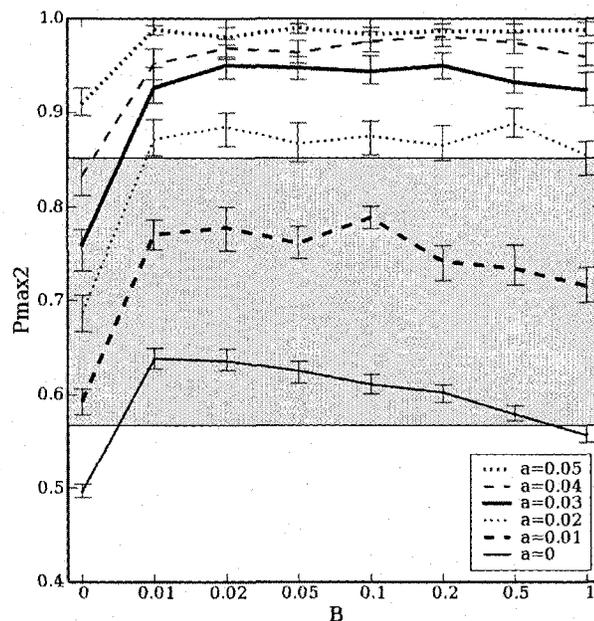


Figure 8.6: Effects of varying both a and B

The graph shown in Figure 8.6 is somewhat difficult to interpret. Each line shows the

behaviour of a set of models with the same a value, but different B values. For example, the thick dashed line (second from the bottom) indicates that all of the models with an a value of 0.01 result in models which are within the human confidence interval, no matter what value of B is used. The bottom line indicates that if a is 0, then B must be greater than zero and less than 1 for models that match well to the human data. At the top of the graph, it can be seen that models with a higher a (0.02 up to 0.04) are only good if B is zero.

To aid interpretation, these values are re-presented in Figure 8.7 as a *contour plot*. Here, the x-axis indicates the B value used, and the y-axis indicates the a value. The performance of the model (as measured by the equivalence test threshold) is indicated by shading. The darker the shading, the larger the difference between the model and reality. The dark line indicates the size of the human confidence interval for this measure. What this means is that any combination of parameter values that is *inside* the area marked by the black line (i.e. the lighter portion of the graph which runs along the lower third of Figure 8.7, with a bend upwards on the left) is a good model. These values produce models whose confidence intervals are entirely within the human confidence interval. This allows for a quick identification of good parameter values. This can also be seen as a two-dimensional version of the equivalence test threshold plots given in Figure 8.3 and the right half of Figure 8.4. Importantly, identifying good models using this contour map approach gives the same set of good models as identified in Figure 8.6. Due to its clarity, the contour map will be used in general for the rest of this thesis whenever two (or more) parameters are being varied.

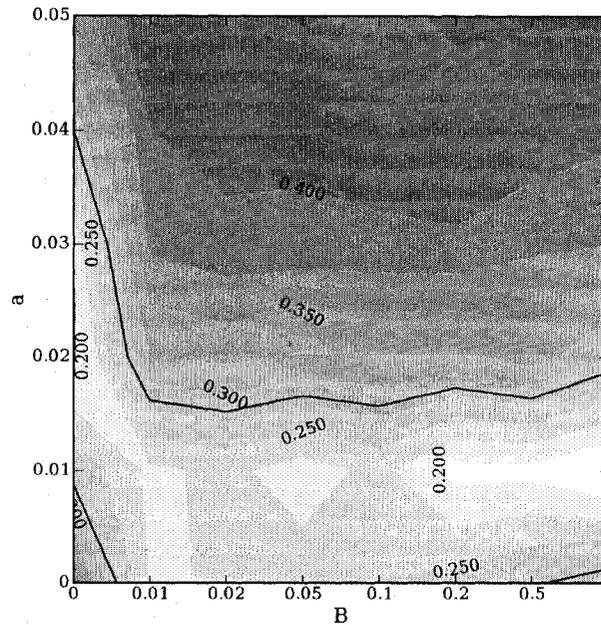


Figure 8.7: Contour plot of the effects of changing a and B . All of the parameter values in the light region bound by the black line give good models.

When examining these contour plots, it is important to remember that they do make use of *linear interpolation* between the data points. The only values that are experimentally known are those for the particular models that were tested. For example, in the above plot it can be seen that the model with $a=0.01$ and $B=0.01$ is a good model, as it is lightly coloured and inside the thick line). It can also be seen that the model with $a=0.02$ and $B=0.01$ is not a good model, as it darkly coloured (and on the other side of the thick line). Looking at the graph, it seems as though a model with a value of a somewhere around 0.017 and a B of 0.01 would be right on the border line of being good or not, as this is where the thick line appears. However, that particular combination of parameters *has not been run*. All that is actually known is that $a=0.01$ and $B=0.01$ gives a good model and $a=0.02$ and $B=0.01$ gives a poor model (i.e. a model which is less accurate than we could

hope for given the human confidence interval). If the values are linearly interpolated (i.e. if a straight line is drawn between these data points, as in Figure 8.2 or Figure 8.6), then the transition from inside the confidence interval to outside is somewhere around 0.017. This will only be true if the model's performance is perfectly linear. Since this is seldom the case, these intermediate values (i.e. values that have not been directly tested) should not be interpreted as definite conclusions. Instead, they are merely meant to give an overall indication of how the behaviour of the model might be reasonably expected to change. This sort of assumption is required as it is impossible to run the simulations with every possible value. Of course, the more values that are tried, the more accurate the graph will be.

8.3 Changing Three or More Parameters

For more than two parameters changing at the same time, it is simply necessary to run even more simulations. To interpret the results, a collection of contour plots can be created. In the above section the parameter L was fixed at 8. Table 8.4 shows that it will now be varied across 8 possible values, resulting in the 8 contour plots shown in Figure 8.8. This now requires 384 ($8 \times 6 \times 8$) different parameter settings.

L	1	2	4	8	16	32	64	128
a	0	0.01	0.02	0.03	0.04	0.05		
B	0	0.01	0.02	0.05	0.1	0.2	0.5	1
K	4							

Table 8.4: Parameter settings for the RELACS model with two parameters being varied at the same time.

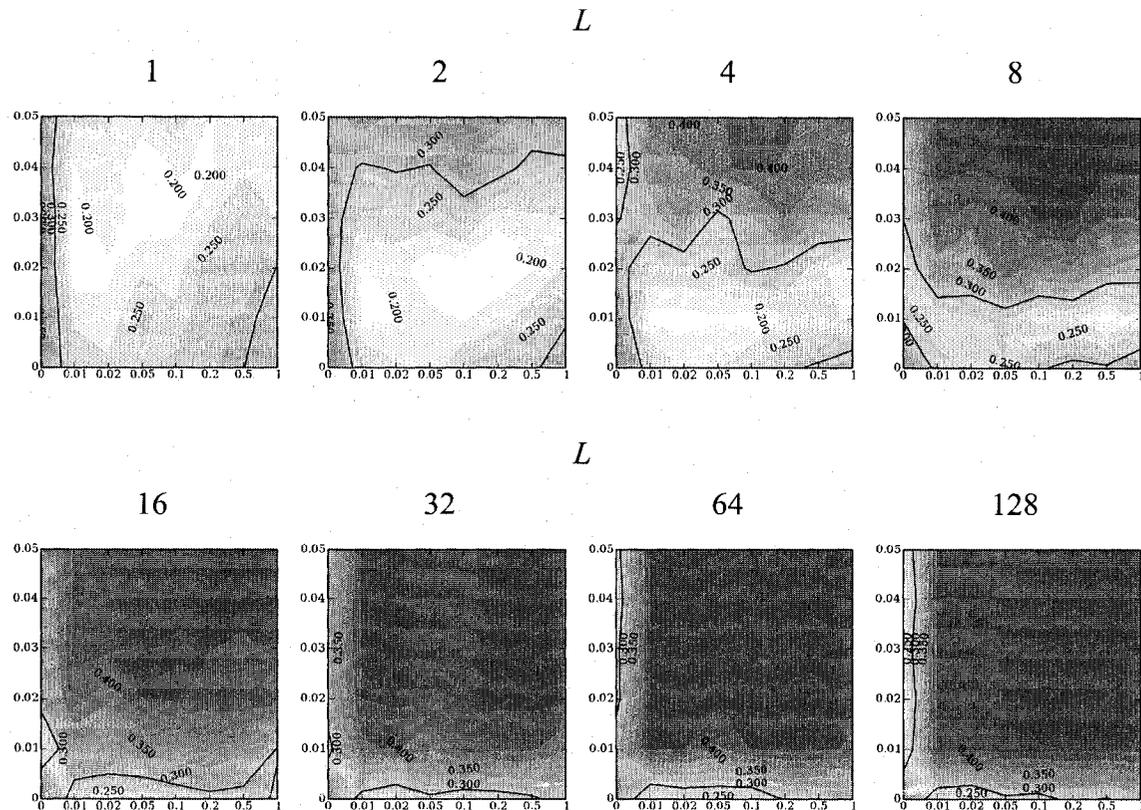
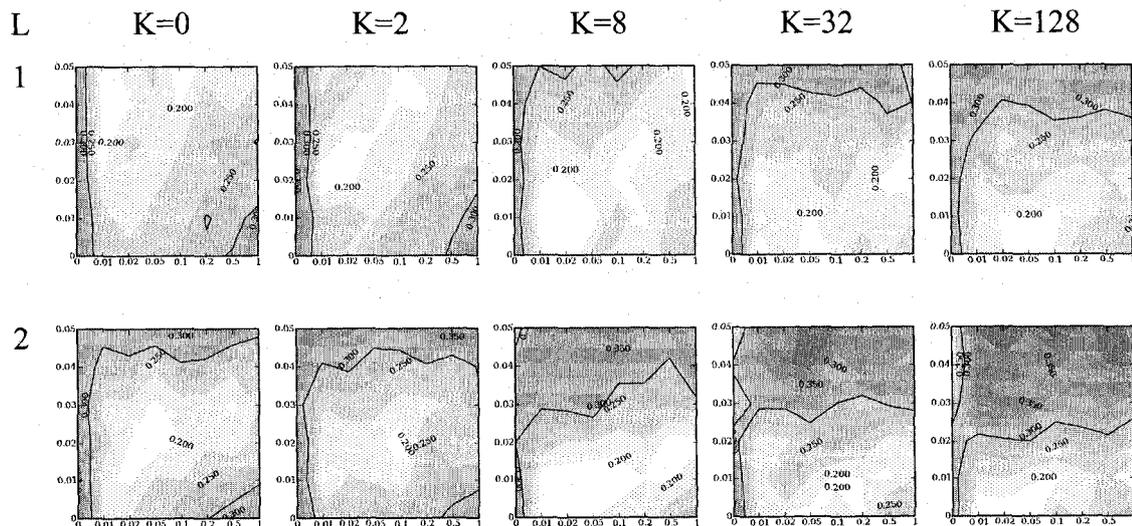


Figure 8.8: Effects of varying a , B , and L . In each plot, the x-axis indicates the B value and the y-axis indicates the a value. Models that are within the human confidence interval range are the lighter coloured areas between the thick line.

The plot for $L=8$ is the same as the one in Figure 8.7, and the others show the effects of adjusting the parameter L over a wide range. This reveals that there are a wide range of models which potentially match the human performance. On this measure, a model with $L=128$, $a=0.04$, and $B=0$ potentially matches the reality, as does a model with $L=8$, $a=0.01$, and $B=0.02$, or $L=1$, $a=0.03$, and $B=0.5$. Each of these models may give very different predictions, but they all can be confidently said to be within 0.284 of the human data. As noted above, 0.284 is the size of the human confidence interval, and so indicates how uncertain the human data values are. Without more detailed real-world data, there is no way to establish any of these models as a better match than another.

It is, of course, possible to vary a fourth parameter as well. In all of the above cases, the K parameter remained fixed at 4. Examining the effects of this change requires an even larger number of models to be run. Fortunately, in this case the parameter K turns out to have a very weak effect on the overall performance of the model. Instead, the general pattern is that low values of L (1 or 2) generally produce good models, and at very high values of L (64 or 128) either a or B must be zero to produce a good model. The only effect K has is that when L is 1, a high K (32 or 128) requires an a below 0.03 to give a good model. Describing this overall effect in words is difficult, so Figure 8.9 shows the complete effect. All light-coloured areas bounded by a thick line indicate parameters which produce models which are guaranteed to be no farther away from the actual human performance than the size of the human confidence interval. None of these models can be distinguished from each other without more empirical data to compare to.



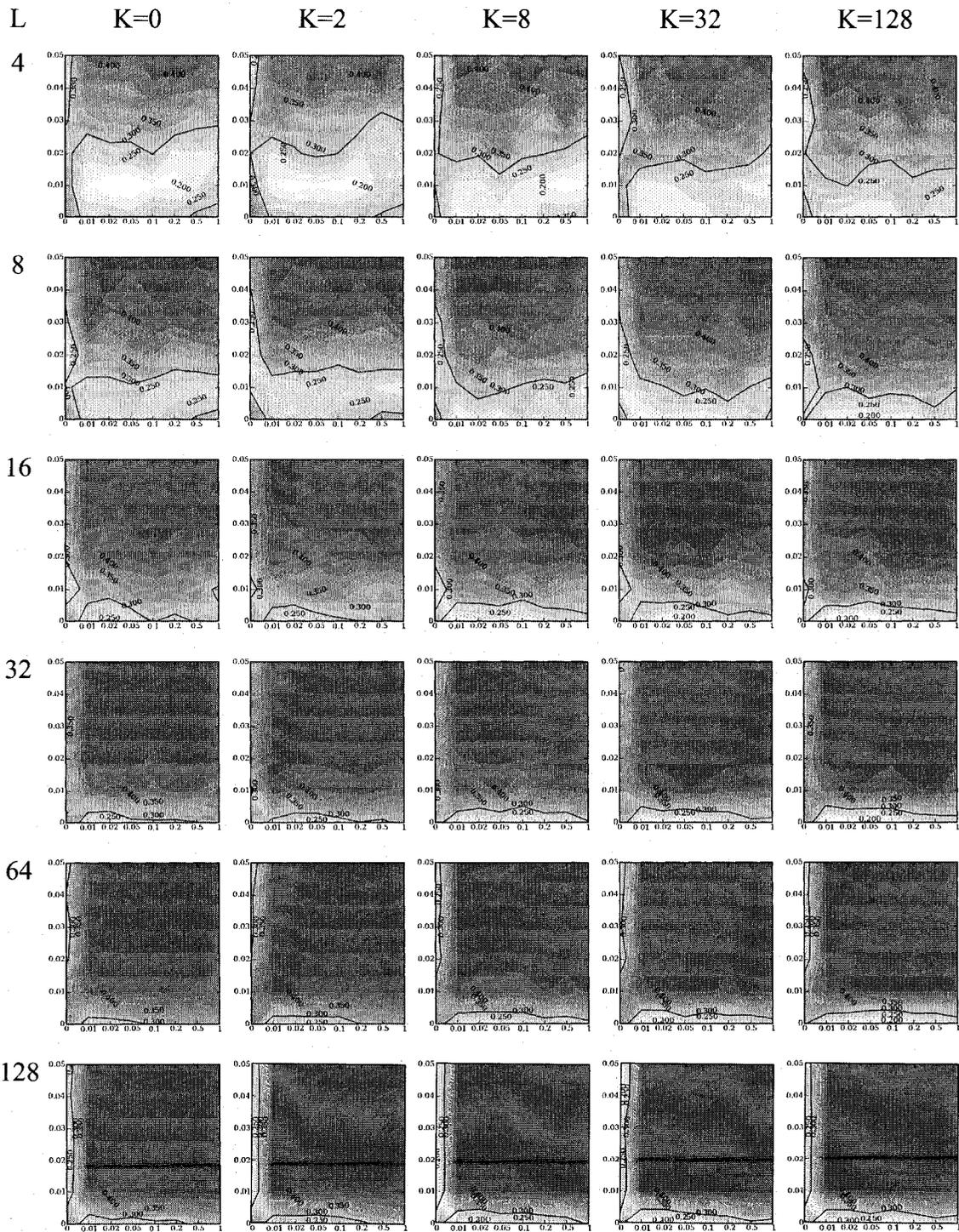
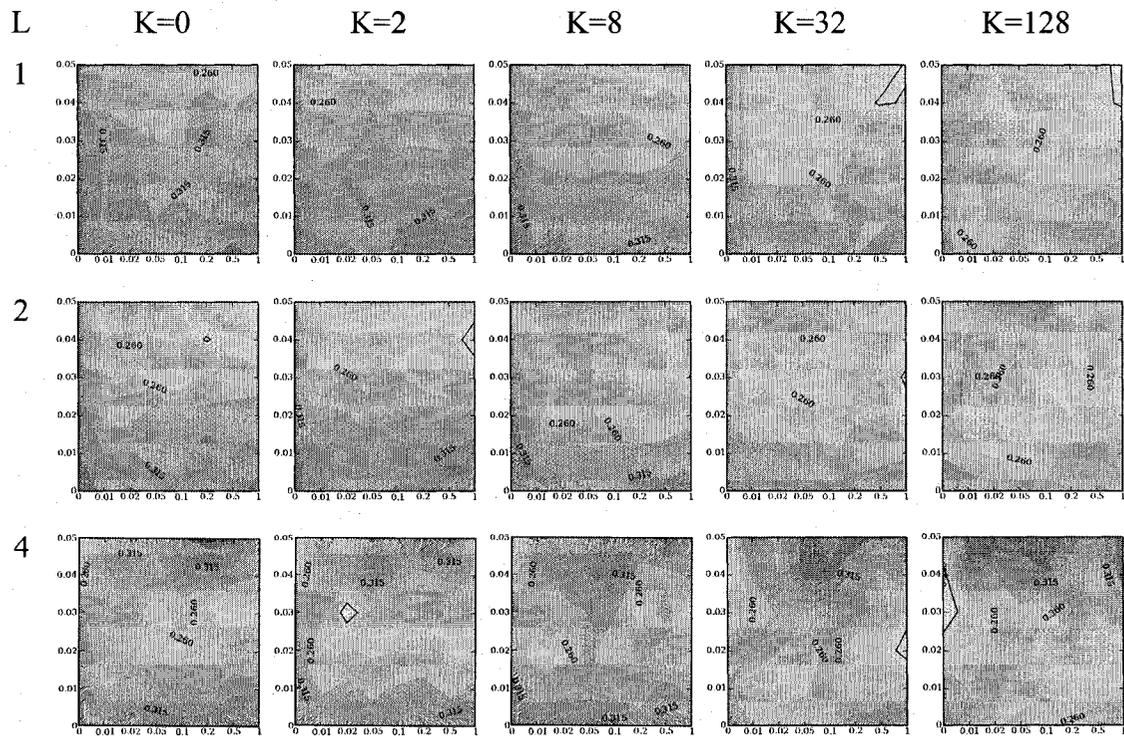


Figure 8.9: The effects of varying all four parameters

As a further complication, it is important to realize that all of the above analysis was only concerned with the *mean* performance of the model. For a model to capture the real

world situation, it must not only give the same mean response, but should also vary in a similar way. What is desired is that the *distribution* of the model be the same as the distribution of the real data. To confirm this, statistics other than the mean must be considered.

Since the only distributional information available from the human subjects in this case is the standard deviation, this is the only measure other than the mean which can be used. As could be seen in Figure 7.3, above, the RELACS model at its default parameters has a much smaller standard deviation in its performance than is seen in the human participants. This is also the case if the parameters are allowed to vary, as can be seen in the following figure.



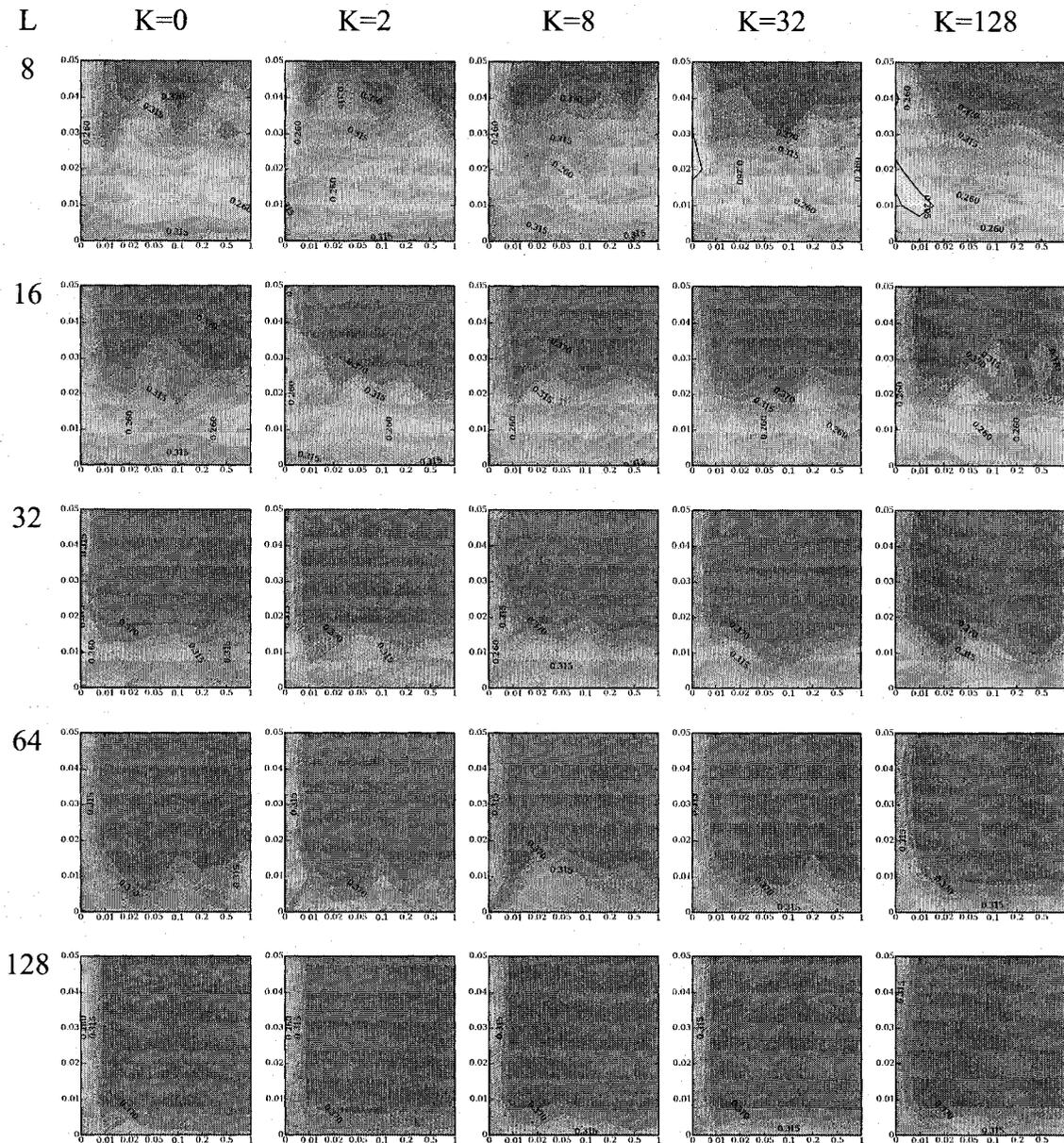


Figure 8.10: The effects of varying all four parameters on the standard deviation of Pmax2

Looking at the above figure, it can be seen that while a large number of RELACS models could be suitable for modelling human performance in terms of the *mean* Pmax2 measurement in Experiment 2 by Erev and Barron (2005), very few models match in terms of the standard deviation. Furthermore, most of the ones that do match in terms of the standard deviation (i.e. the models with parameter values that are inside the thick

lines in the above graphs) end up not matching in terms of their mean performance (as seen in Figure 8.9, above). The only exceptions are the two models identified in Table 8.5.

	Model 1	Model 2	Human Data
<i>a</i>	0.04	0.04	
<i>B</i>	0.2	1	
<i>L</i>	2	2	
<i>K</i>	0	2	
Pmax2 mean	0.714 – 0.831	0.709 – 0.856	0.568 – 0.852
Pmax2 std dev.	0.182 – 0.236	0.196 – 0.239	0.178 – 0.396

Table 8.5: The only two RELACS models known to differ from the human performance by less than its confidence interval for both the mean and standard deviation. These models are statistically indistinguishable from the real data.

As noted above, the RELACS model is known to have much less variability in behaviour than is seen in the human participants. With this in mind, it not surprising that few models match in terms of both the mean and the standard deviation of the behaviour. It should be noted that in the original paper (Erev & Barron, 2005), the parameters were explored in terms of the *mean* performance only, and only a small section at the end noted that the model exhibited much less variability. However, the only model that was examined for its variability was *the one model that best fit the mean data*. Figure 8.10, above, shows that the standard deviation of the performance of the RELACS model varies considerably based on its particular parameter settings. This means that if one is attempting to develop models which exhibit behaviour that has the same distribution as human performance, one cannot focus solely on the mean, as is almost universally found in modelling work. Instead, these various statistical aspects of the behaviour distribution

(such as the standard deviation, and even the skew, kurtosis, median, and any others) can also be considered.

However, if more than one statistic is to be considered at the same time, there needs to be a way to *combine* these into a single measurement. This will be examined in the next chapter.

8.4 Summary

8.4.1 Methodology

This chapter has addressed investigating multiple models in a single situation with a single measure. These models were all related to each other by having the same architecture, but varying parameter values. Up to four parameter values were adjusted. This identified a large set of parameter settings all of which result in models that are equivalent to the human data. All of these models must be treated as equally plausible explanations of the human data. Even though some of them may have a closer numerical fit to the human data, this closer fit is a symptom of over-fitting to the particular subjects' data, not anything about the population as a whole.

Since there are so many equivalent models, the methodology suggests that more measures be investigated. This is done next, in chapter 9.

8.4.2 Contributions

- A large set of RELACS parameter settings were found to produce models which are equivalent to the human data for Erev and Barron's condition 2. Low values

of L (1 or 2) tend to lead to equivalent models, and at very high values of L (64 or 128) either a or B must be zero to produce an equivalent model. However, very few parameter settings are equivalent in terms of the standard deviation.

9 Multiple Measures

All of the previous chapters discussing the application of the computational cognitive modelling methodology have focused on a single measurement, Pmax2 (the proportion of time in the second block of 100 trials that the choice with the maximum expected value was made). This single measurement was only made in one situation, where pressing one button always gave a reward of 11, and pressing the other button would give a reward of 19 half of the time, and 1 the other half of the time (Experiment 2 from Erev & Barron, 2005). In other words, so far RELACS has only been considered as a model for that one situation, and only one aspect of its performance has been examined.

There are, of course, other measures and other situations that can be examined. The only limitation is the availability of raw human data for comparison. These other measurements can be examined in exactly the same way as the single measurement was in the previous chapters: the model is run multiple times in each situation, with multiple parameter values, and the overall performance on each measure can be determined.

The only new complication is how to *combine* these measures into a single value which represents the overall performance of the model on all of the different measures. For example, if a model is within 0.2 on measure A, 0.3 on measure B, and 0.1 on measure C, then what overall value can be given to it? As discussed previously in section 4.2, it can be tempting to simply calculate the *average* value (in this case, 0.2) and use that as an indication of the overall performance of the model. However, this average value can be misleading. Instead, the way the values will be combined here is to take the *maximum*

error. That is, if the model is known to be within 0.2 on A, 0.3 on B, and 0.1 on C, then the overall conclusion should be that it always produces behaviour within 0.3 of the real-world data, on any of the three measures.

However, this approach for combining values only works for measures which are in equivalent units. If there are not in equivalent units, then it is difficult to determine whether a difference of 100 milliseconds is larger or smaller than a difference of 20% accuracy. Furthermore, this simple comparison by taking the maximum also works best when the measures have similarly-sized confidence intervals. For example, if measure B has a very wide confidence interval on the human data (compared to measure C), then it is unsurprising that the models would have a larger maximum difference for this measure.

For these situations, the relativized equivalence can be determined. Here, the equivalence test measure is divided by the size of the real-world confidence interval.

This means that models with a value of 1.0 or less differ from the real world by less than the real world confidence interval, and those with a greater value differ by more. This relativized value can be used when combining measures. For example, if the largest of these relative equivalence measures is 0.9, then all of the measures being considered have values which are indistinguishable from the human performance. If the value is 1.1, then at least one of those values may be distinguishable, but the difference is not as great as a model with a value of 1.2. This approach was initially described in section 4.2.8, and will now be demonstrated in two different cognitive domains.

9.1 Multiple Measures: Repeated Binary Choice

For a first example of using multiple measures, the repeated binary choice task and the RELACS model will be used in an extension of the work described in the previous chapters. Instead of the single situation used previously, the following three experimental situations will be considered.

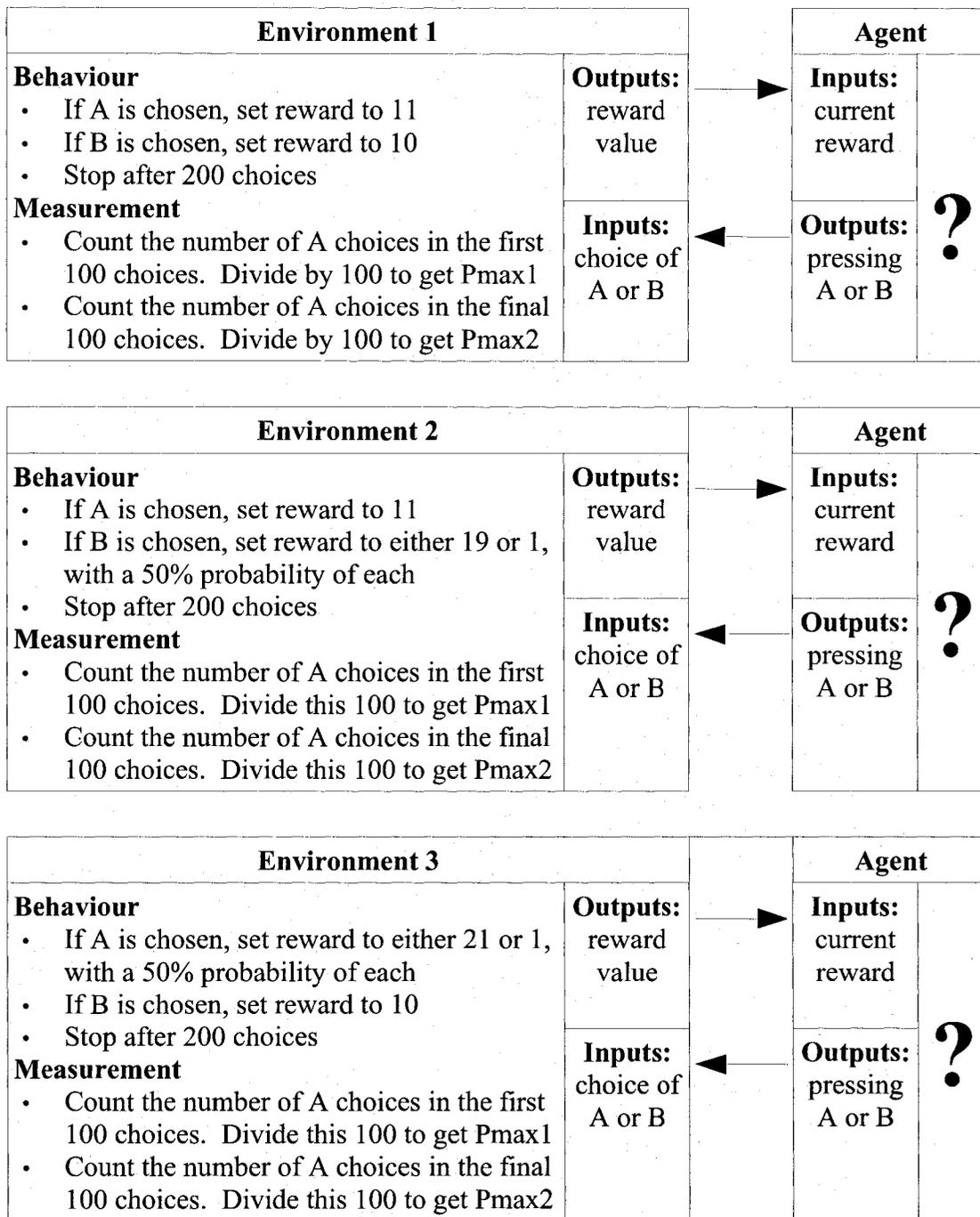


Figure 9.1: Agent-Environment interaction and environment specification for experiments 1,2, and 3 (Erev & Barron, 2005).

This provides three different experimental situations, and two measures (Pmax1 and Pmax2) in each. The reason for choosing these three experiments is explicitly given by Erev and Barron (2005), as this represents an exploration of risk-seeking and risk-

aversion behaviour. Given the experimental human data in Table 9.1, they note that higher rate of choosing A in experiment 1 over experiment 2 seems to indicate risk-seeking behaviour. This is because option B is chosen more in experiment 2, and the only difference is that in experiment 2 option B had the same expected value (10), but a greater payoff variability. However, when comparing experiments 1 and 3, risk-aversion is seen, as option A is chosen less often, although its expected value is the same (11) while its variability is increased.³¹ Although risk-aversion and risk-seeking are generally treated in psychology as individual factors, this set of experiments shows that these sorts of phenomena can arise via the manipulation of more basic environmental factors. These three experiments thus demonstrate a basic phenomenon that any model of repeated binary choice in humans should replicate.

The complete human data points for these experiments are not, unfortunately, available. However, the means and standard deviations of the Pmax2 values can be found in the paper. To complicate matters, the paper contradicts itself, giving values of 0.90 and 0.87 for Pmax2 in experiment 1, and similar differences for experiments 2 and 3. The values used here are the ones which seem to best correspond to the given graphs. For Pmax1, mean values must be read off of the graphs, introducing some inaccuracy. Furthermore, the Pmax1 standard deviations are not available, and so are set to the same values as the Pmax2 standard deviations. This is clearly not correct, but no other options are available if these values are to be used.

³¹ Although it is not mentioned in the (Erev & Barron, 2005) paper, it is worth noting that while the difference between experiment 1 and 3 is significant at the 95% confidence level, the differences between experiment 1 and 2 is not (it is only at the 90% confidence level).

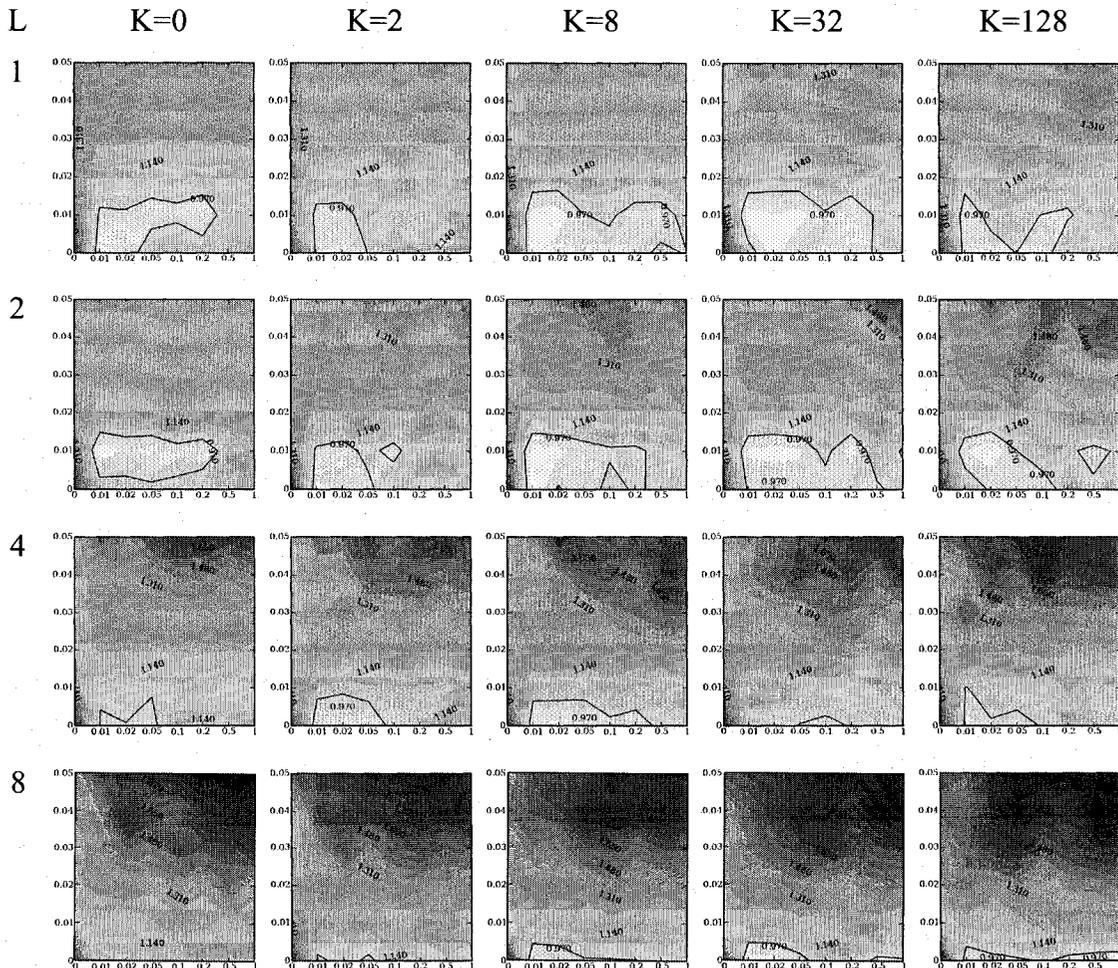
Environment	Measurement	Sample Mean	Sample Standard Deviation
1	Pmax1	0.82 (estimated from graph)	0.152 (assumed)
	Pmax2	0.87	0.152
2	Pmax1	0.64 (estimated from graph)	0.246 (assumed)
	Pmax2	0.74	0.246
3	Pmax1	0.55 (estimated from graph)	0.310 (assumed)
	Pmax2	0.59	0.310

Table 9.1: Data from experiments 1, 2, and 3 (Erev & Barron, 2005). In all cases, N is 14.

Given these results for comparison, the RELACS model can be evaluated over a range of parameter settings. Given the same ranges as appeared in Figure 8.9 above, the simulation must be run multiple times for 1,728 different parameter settings. For each of these settings, the equivalence difference between the model and reality is determined, and then divided by the size of the real-world confidence interval to give the relativized equivalence measure.

Examining the data, it is immediately obvious that the RELACS model consistently under-predicts the real-world standard deviation. If the standard deviations of the six measures are included in evaluating these models, then none of the RELACS models give results that are statistically indistinguishable from the real-world behaviour. This can be interpreted as a falsification of this model, although it is still technically possible that some unexamined parameter values for the model would produce suitable results. However, since there is no good way to find such values other than a more exhaustive search of the parameter space, for now it can be concluded that RELACS does not produce an equivalent performance distribution as human subjects on these two measures in these three situations.

However, it is unsurprising that RELACS produces less variable behaviour than the real subjects, as this aspect was specifically ignored by the developers of the model. With this in mind, it may be useful to only examine the *mean* performance of the model, and ignore the standard deviations. These results are shown in Figure 9.2. The graphed value is the maximum of the relativized equivalence test thresholds. In other words, all of the models in the light area within the thick line are known to be statistically indistinguishable from the real data.



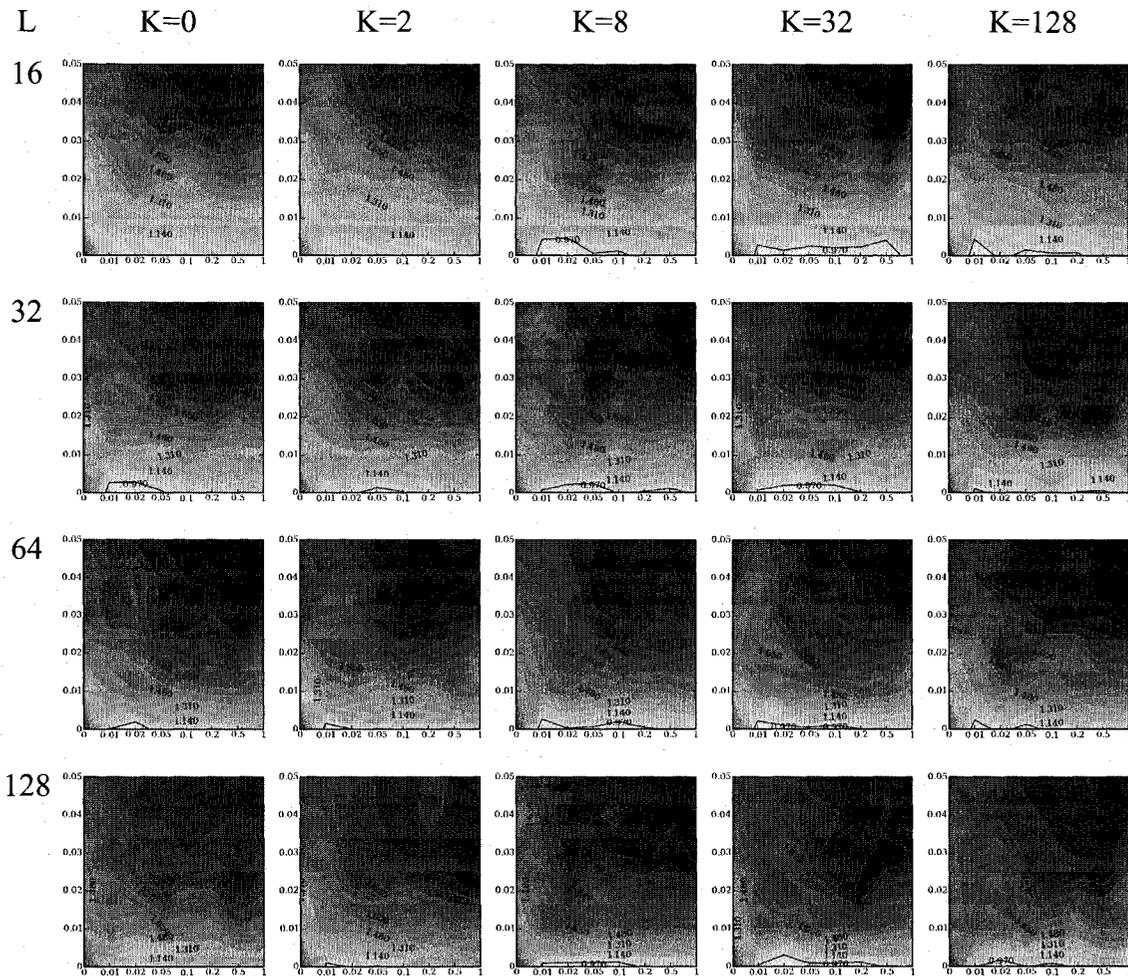


Figure 9.2: The effects of varying all four parameters on the mean of P_{max1} and P_{max2} in environments 1, 2, and 3. A value of 1.0 (the thick black line) indicates that all measures differ from the real world value by no more than the size of the confidence interval. Thus all models inside the thick black line are statistically indistinguishable from the real world data.

This can be seen as a refinement of the results in Figure 8.9. Adding five more measures reduces the set of suitable models considerably. Instead of almost any model with an L of 1 matching the human data, now models must have an a of 0 or 0.01. K continues to have little effect overall. The potential explanatory models identified by this process can be roughly divided into two groups. The first group is those with a low L (1 to 4), which have an a of 0 or 0.01, and any K or B value. The second group is seen with a much higher L (8 to 128), and must have an a of 0, a K of 8 or larger, and a B value between

0.01 and 0.5. All of these models might be seen as possible explanations of the human mean performance on this set of three experimental situations.

9.1.1 Many More Situations

This approach can also be extended to large numbers of additional measures. Erev and Barron (2005) provide data for 40 different situations. Their analysis involves finding the one model with the smallest Mean Squared Difference from the human data. As discussed in section 4.2.2, this approach is prone to over-fitting the data, and can only lead to an indication of one model that happens to be the best statistical predictor, given the particular random sampling of individuals chosen for the experiment. Instead, if this equivalence test metric is used, a range of possible models can be identified which could be matches for all of the human data.

The 40 different conditions come from a variety of studies, most of which were conducted by Erev and his colleagues. Precise details on all these conditions can be found in their paper. Two conditions (#29 and #30) had to be removed from consideration, since no information was available on their standard deviations, making it impossible to estimate confidence intervals. Also, standard deviation information was only available for the 2nd block of 100 trials in each condition. The one exception to this is conditions #15 to #20, for which only the 4th block is available from the original source (Myers et al., 1963).

The remaining conditions (summarized in Tables 9.2, 9.3, 9.4, and 9.5) are divided into four categories, based on the effects being demonstrated. Most of the first 22 conditions

(with the exception of #15 to #20) demonstrate variations on the Payoff Variability Effect. This involves adjusting the variation in the outcomes for a given choice, without adjusting the mean outcome. Observed behaviour changes from risk-seeking to risk-aversion depending on whether the variability is associated with the overall best option

Conditions #15 to #20 examine changing the magnitude of the reward. Here, choice A is correct 60%, 70%, or 80% of the time, and the reward is 1 or 10. This translates into different monetary rewards given to the participants.

Conditions #23 to #25 investigate the under-weighting of rare outcomes. Here, events that are very rare (<10%) seem to not be considered when determining expected outcomes.

The remaining 15 conditions (#26 to #40) deal with the Loss Rate Effect. Here, “when the action that maximizes expected value increases the probability of losses, people tend to avoid it” (Erev & Barron, 2005, p. 917). That is, a choice that has a higher expected value in the long run may be chosen less often if it is comprised of many small losses and few large gains.

#	Reward A	Reward B	
1	11	10	
2	11	50%: 19; 50%: 1	
3	50%: 21; 50%: 1	10	
4	-10	-11	
5	-10	50%: -21; 50%: -1	
6	50%: -19; 50%: -1	-11	
7	11	10	(reward shown for A and B)
8	50%: 21; 50%: 1	10	(reward shown for A and B)
9	-10	-11	(reward shown for A and B)
10	-10	50%: -21; 50%: -1	(reward shown for A and B)
11	G(21,3)	G(18,3)	
12	G(21,3)	G(18,3)+(50%: 5; 50%:-5)	
13	G(21,3)+(50%: 5; 50%:-5)	G(18,3)	
14	G(21,3)+(50%: 5; 50%:-5)	G(18,3)+(50%: 5; 50%:-5)	
21	80%: 4; 20%: 0	3	
22	20%: 4; 80%: 0	25%: 3; 0%: 0	

Table 9.2: Experimental conditions that exhibit the payoff variability effect. $G(\mu, \sigma)$ indicates a normal Gaussian distribution with mean μ and standard deviation σ .

#	Reward A	Reward B
15	60%: 1; 40%: -1	40%: 1; 60%: -1
16	60%: 10; 40%: -10	40%: 10; 60%: -10
17	70%: 1; 30%: -1	30%: 1; 70%: -1
18	70%: 10; 30%: -10	30%: 10; 70%: -10
19	80%: 1; 20%: -1	20%: 1; 80%: -1
20	80%: 10; 20%: -10	20%: 10; 80%: -10

Table 9.3: Experimental conditions that show the effects of adjusting the magnitude of the reward.

#	Reward A	Reward B
23	10%: 32; 90%: 0	3
24	2.5%: 32; 97.5%: 0	25%: 3; 75%: 0
25	-3	10%: -32; 90%: 0

Table 9.4: Experimental conditions that show the underweighting of rare outcomes effect.

#	Reward A	Reward B	
26	G(100,354)	G(25,17.7)	(reward fixed to be ≥ 0)
27	G(1300,354)	G(1225,17.7)	
28	G(1300,17.7)	G(1225,17.7)	
31	70%: 6; 30%: 2	30%: 6; 70%: 2	
32	70%: 4; 30%: 0	30%: 4; 70%: 0	
33	70%: 2; 30%: -2	30%: 2; 70%: -2	
34	70%: 0; 30%: -4	30%: 0; 70%: -4	
35	70%: -2; 30%: -6	30%: -2; 70%: -6	
36	63%: 6; 27%: 2; 10%: 0	27%: 6; 63%: 2; 10%: 0	
37	63%: -2; 27%: -6; 10%: 0	27%: -2; 63%: -6; 10%: 0	
38	63%: 6; 27%: 2; 10%: 0	27%: 6; 63%: 2; 10%: 0	(only show win/loss, not reward)
39	63%: -2; 27%: -6; 10%: 0	27%: -2; 63%: -6; 10%: 0	(only show win/loss, not reward)
40	-3	80%: -4; 20%: 0	(for comparison with #21)

Table 9.5: Experimental conditions that show the loss rate effect.

This results in four groups of measures which can be examined individually or all together. The first to be examined is the payoff variability effect (conditions #1 to #22, excluding #15 to #20). If all of these measures are combined, then the result is that *no parameter setting leads to a successful model*. That is, there is no RELACS model that was found to be equivalent to the real human data on all of these measures.³² At most 14 conditions are matched; in these cases, #1, #7, #11, and #13 are the most problematic. Conditions #1 and #7 are the simplest cases (A always giving a reward of 11 and B always giving 10), while #11 and #13 are the most complex, giving real-value rewards (with a Gaussian distribution), as opposed to a one of a small fixed number of rewards. In other words, RELACS is *incapable* of accounting for these cases while still accounting for the rest of the payoff variability effect. RELACS is thus an inappropriate model for such cases.

³² It is, of course, possible that there is a good RELACS model for these measures, but that it requires parameter settings that were not tried. This is always a possibility for any model, but this parameter exploration did not find such a situation.

If these conditions are eliminated from consideration, many parameter settings fit the remaining conditions. These can be seen in Figure 9.3. Here, models are equivalent to the human data mostly when L and K are small, or when L is large (~ 128) and a is zero. This is a slightly more restricted region than what was seen in Figure 9.2, which only used three of these measures.

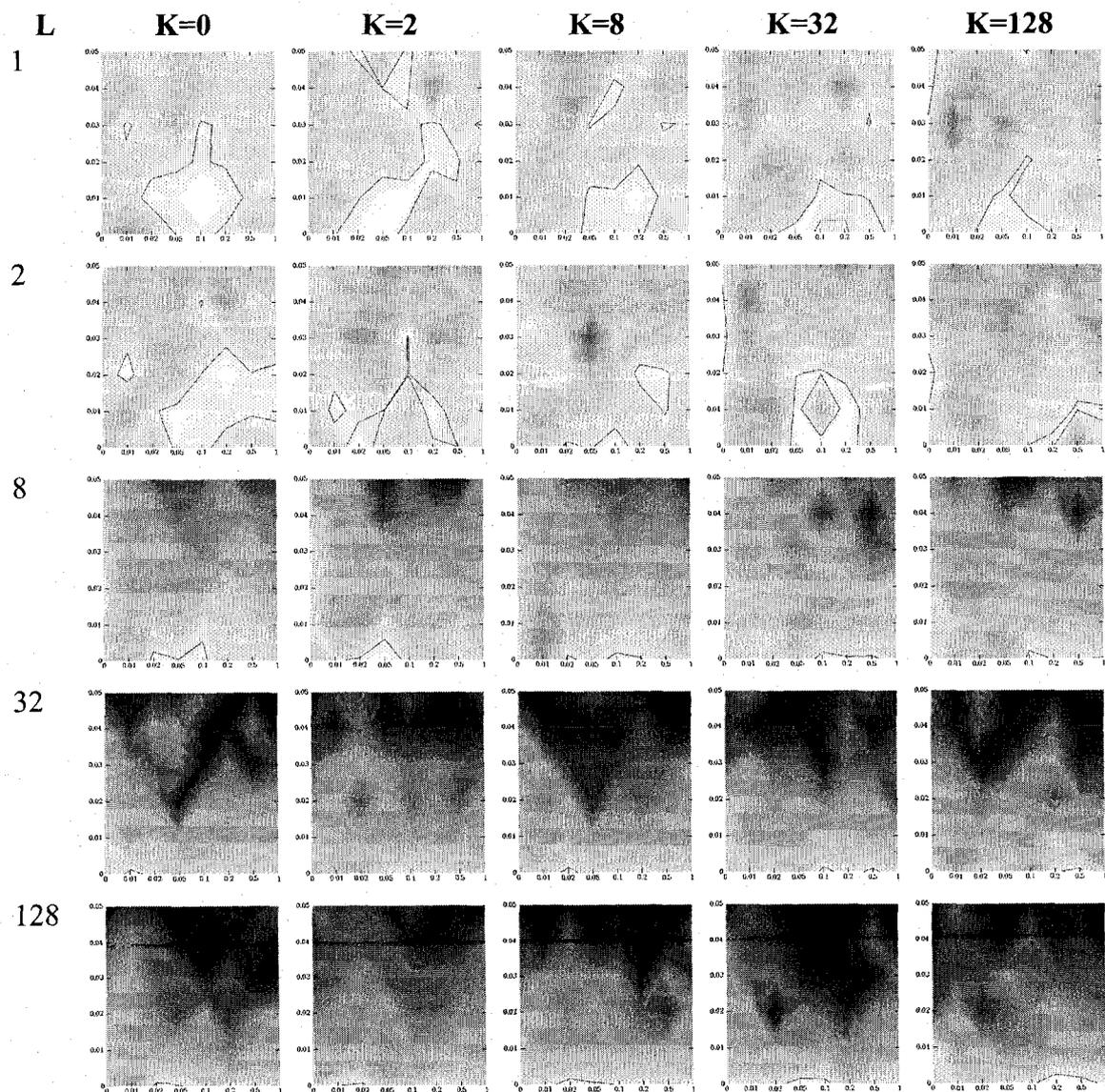


Figure 9.3: Results for the Payoff Variability Effect measures, excluding measures #1, #7, #11, and #13. If those four are included, then no models are equivalent to the human data.

The six conditions where reward is adjusted (#15 to #20) are not graphed here. This is because the RELACS model turns out to not change its behaviour at all when reward values are scaled. For confirmation, the original paper (Erev & Barron, 2005, Figure 2, p. 915) reveals identical model performance, but changing human data. This fact is not commented on in that paper. It is clear, then, that RELACS cannot account for this aspect of human performance. It should be noted that that the MSD approach to analysis used by Erev and Barron did not highlight this fact.

A wide selection of parameter settings are equivalent in terms of the three measures for underweighting (#23 to #25). These are shown in Figure 9.4. It can be seen that almost every model with $a=0$ is equivalent in these conditions, and that particular values for K and B have relatively little impact.

Next, Figure 9.5 shows the results for the Loss Rate Effect measures. As with the payoff variability conditions (Figure 9.3), there were no parameter settings that were equivalent on all 13 loss rate effect conditions. However, if #28, #33, #34, and #40 were removed, then equivalent models are found for the remaining conditions. #28 has a Gaussian reward structure (much like #11-#14), so it is expected that RELACS will fail here. #33 and #34 deal with a change in magnitude of reward, so can be eliminated much like #15-#20, as RELACS simply cannot account for this.

Condition #40, however, is very similar to the other conditions which are successfully modelled, and there seems to be no good reason to remove it. However, it must be removed or no good models can be found. There are two possible interpretations here.

One is that RELACS simply cannot account for this range of conditions. Another possibility which can be raised due to this surprising failure of RELACS (after all, it successfully models many situations similar to #40) is that the empirical results themselves are in error. Further experimentation is needed to determine this.

If all of these measures are combined together, it is possible to identify parameter settings for which RELACS gives equivalent results in all of these conditions. These are shown in Figure 9.6. As can be seen, only a very few parameter settings provide equivalent results. The most obvious one is at $a=0.01$, $B=0.05$, $L=1$, and $K=0$. There are also a few with $a=0$ and large L . These represent the RELACS models which are the equivalent to the human data on the greatest number of measures. This includes all of the original measures, excluding the following conditions:

- #1, #4, #7, #9: where the rewards for A and B are fixed (e.g., one always gives a reward of 10 while the other always gives a reward of 11).
- #11-#14, #26-#28: where the rewards are given by Gaussian distributions
- #15-#20, #33, #34: where the only changes are the magnitudes of the rewards, which RELACS does not account for
- #40: an outlier which must be removed to attain equivalent models

The conclusion that can be drawn from this is that RELACS is only successful at modelling situations where the rewards are relatively simple. All of the situations that it successfully models deal with rewards that only take on a few different values. For example, in condition #24, A gives a reward of 32 2.5% of the time (and zero the rest of the time), while B gives a reward of 3 25% of the time (otherwise, zero). If rewards drawn from a probability distribution instead, then RELACS does not successfully model

the results³³. If the rewards are too simple (i.e. if both of the rewards are fixed values), then RELACS also fails. It also does not respond differently in situations where the only change is the absolute magnitude of the reward.

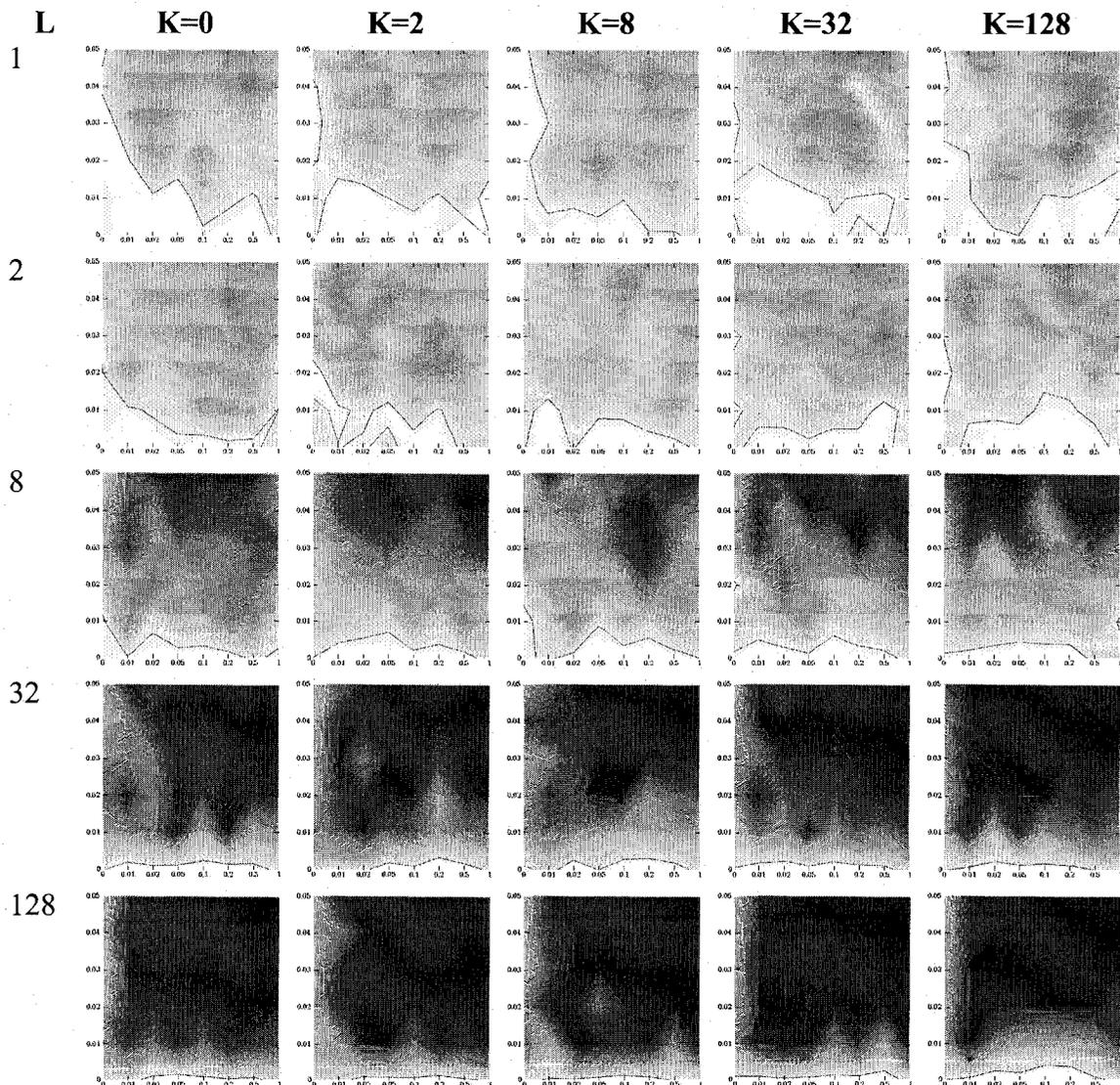


Figure 9.4: Results for the Underweighting Effect measures

³³ More correctly, RELACS cannot handle both the complex probability distributions and the more simple situations with the same parameter settings. A future addition to RELACS may adjust parameter settings based on these situations, but that is not part of the current RELACS model.

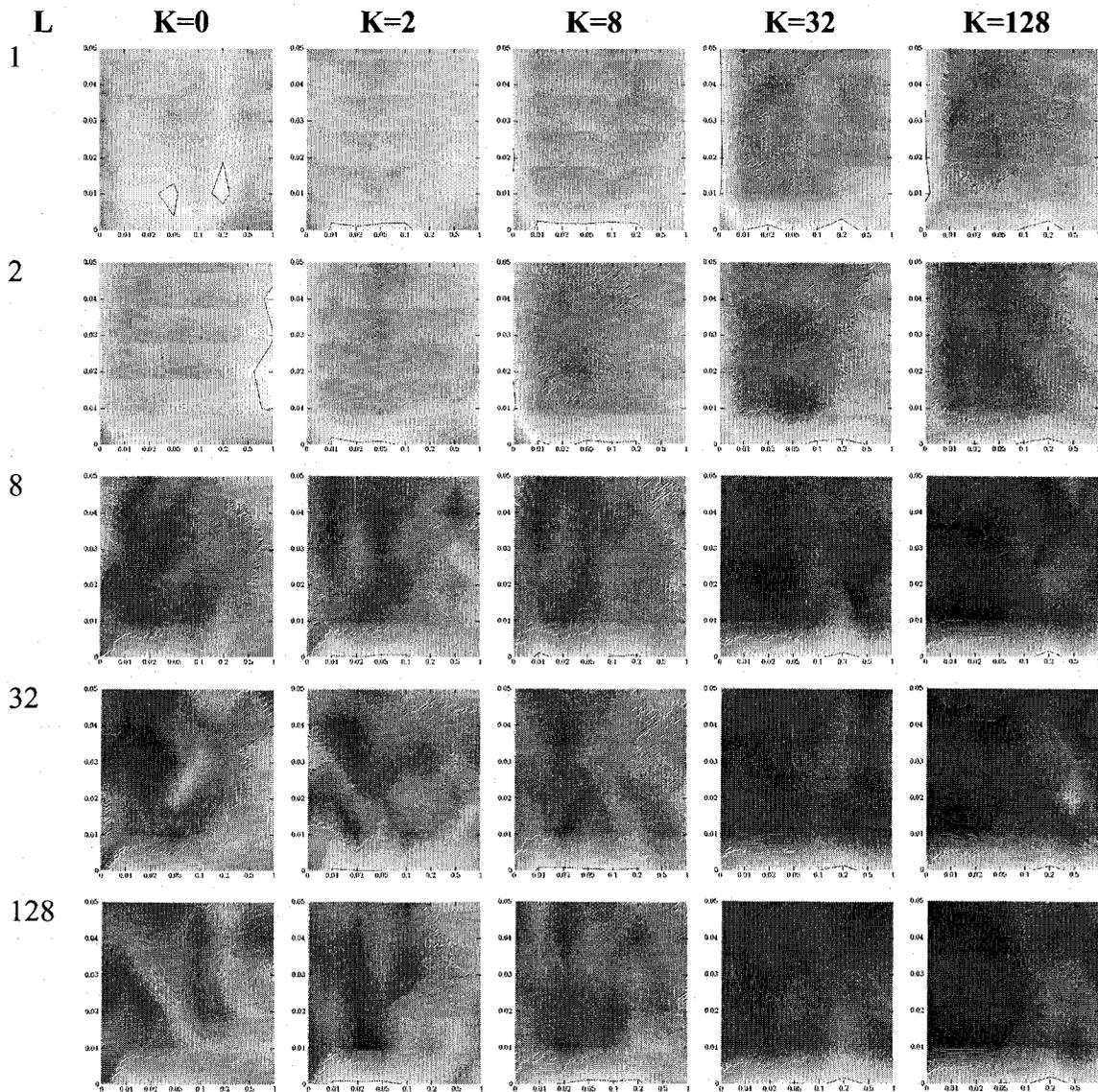


Figure 9.5: Results for the Loss Rate Effect measures, excluding measures #28, #33, #34, and #40. If those four are included, then no models are equivalent to the human data.

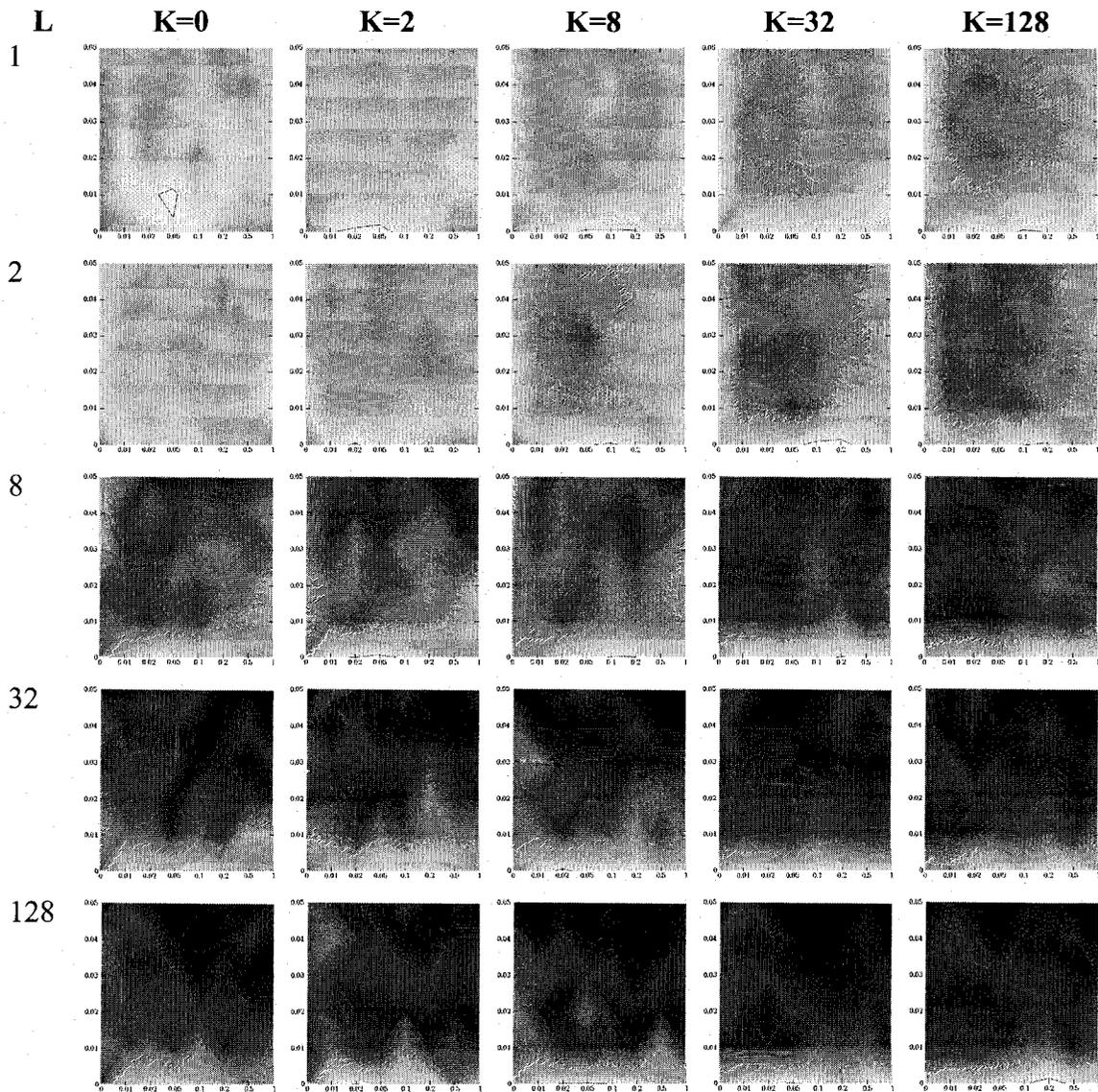


Figure 9.6: Results for the all measures found in Figure 9.3, Figure 9.4, and Figure 9.5. Any models inside the black outline are equivalent to the human results on all considered measures.

By examining these measures via an equivalence test, a very different conclusion has been reached than in the original RELACS paper. There, since the results from all of these measures was averaged together via the Mean Squared Difference, the only conclusion was that the best version of RELACS gave results that were, on average, within 6% of the human data. Using the equivalence measure and the methodology developed in this thesis, a much richer conclusion is available. In particular, it is now

clear that RELACS is limited to moderately complex reward structures, and cannot be seen as a generic model of human response to arbitrary rewards in a repeated binary choice task.

9.2 Multiple Measures: Modelling Peer Groups

In the previous section, combining multiple measures led to a tightening of the constraints on possible parameter values that lead to plausible models. This reduced the space of possible models from a very large one (all of the models inside the thick lines in Figure 8.9) to a very few possibilities.

The underlying phenomenon here is that adding new measures can *invalidate* a model.

That is, a model which performs well on one measure (or one set of measures) may fail to produce behaviour that matches the real-world data on some other measure, or in some other situation. This indicates that a given model may be limited in its applicability. It may be suitable for predicting error rates, but not for reaction times.

This phenomenon has important implications for *explanation*. If a model is predictive in some situations, but not others, then its explanatory power is limited. It may be that it is only an accident that the model is predictive (i.e. it may follow a different internal process in the situation that it works well in, but give the same final behaviour). It may also be that it is a good model of that phenomenon, but that the other measure or other situation requires some other component to be added to the model. Exploring these possibilities is part of the computational cognitive modelling process.

To demonstrate this, a new modelling situation will be considered. The previous chapters

have dealt with a repeated binary choice task, which is a relatively stereotypical experimental psychology situation. This new example is one from social psychology: the modelling of peer group interactions in children. In particular, the model is meant to portray the dynamic processes of popularity, rejection, and neglect over time. This situation will be explored along with the repeated binary choice situation for the rest of this thesis. These sections are adapted from the original publication (Stewart, West, & Coplan, 2007).

9.2.1 Popularity and Peer Groups

The study of the measurement of children's peer relations (i.e. sociometric classification) begins with Moreno's (1934) research. His work on describing individuals in terms of the attraction and repulsion felt by them toward others and by others toward them spawned a wide variety of measurement techniques, each attempting to develop a useful scale for the investigation of the causes and effects of children's social experiences (see Cillessen & Bukowski, 2000 for a review).

One of the results of this research is that a simple, one-dimensional scale is not sufficient to capture useful information. In particular, it is generally necessary to distinguish at least three categories: individuals who are viewed positively by peers, individuals who are viewed negatively, and individuals who are ignored. To do this, most modern sociometric techniques involve distinguishing two measurement dimensions: preference and impact. A person who is ignored would be one with a low impact, while someone who is actively disliked might have a high impact, but very low preference rating.

The most popular and widely used measurement scheme is known as CDC Classification. It is named after its creators, Coie, Dodge, and Coppotelli (1982), and classifies people into one of five categories: Popular, Rejected, Controversial, Neglected, or Average. To facilitate its use across as wide a range of ages and situations as possible, its methodology is quite simple. Using interviews or questionnaires, each person is asked to name three people in their peer group that they like, and three people that they dislike. These scores are then standardized within class to control for the number of possible nominations received. The simplicity of this measurement is important for measuring popularity in young age groups. Using the survey results, each individual is given an Acceptance score (the total number of times that person is listed by other people as someone they like) and a Rejection score (the number of times they appear on the 'dislike' lists). A Preference value (Acceptance minus Rejection) and an Impact value (Acceptance plus Rejection) are also created, where Preference refers to whether you are more liked or disliked and Impact refers to how much people pay attention to you. Individuals are then classified into the five categories according to the rules shown in Table 9.6.

Category	Rule
Popular	Preference: One standard deviation above the mean Acceptance: Above the mean Rejection: Below the mean
Rejected	Preference: One standard deviation below the mean Acceptance: Below the mean Rejection: Above the mean
Neglected	Impact: One standard deviation below the mean Acceptance: Zero
Controversial	Impact: One standard deviation above the mean Acceptance: Above the mean Rejection: Above the mean
Average	None of the Above

Table 9.6: The decision rules for CDC sociometric classification

Given the wide use of this system, and the accompanying availability of a wide variety of experimental results based on the CDC classification scheme, it is a good choice for the basis of comparison for our computational modelling results.

9.2.2 Real-World Data

Newcomb, Bukowski, & Pattee (1993) provide complete results from nine different studies that used CDC categorization on a total of 2,571 students, ranging from kindergarten to grade 9, in groups of around 30 individuals. Overall, these nine studies give the 95% confidence intervals for the percentage of people in each category shown in Table 9.7. This data set gives the first basis of comparison between model and reality. Any successful model should give results within these ranges.

Popular	Rejected	Neglected	Controversial	Average
7%-32%	12%-26%	0-28%	1.6-16%	5.9-69%

Table 9.7: The 95% confidence intervals indicating the percentage of people in each CDC category

As a more stringent test of a model's abilities, there is also the measure of categorization stability over time. Cillessen, Bukowski, & Haselager (2000) give the results of a meta-study which collected the change in CDC categorization over periods of time ranging from one month to four years. This identifies what percentage of individuals who were put in a particular category at one point in time, remained in that category in a later point in time. Table 9.8 gives the 95% Confidence Interval for this data.

Popular	Rejected	Neglected	Controversial	Average
33-44%	39-49%	20%-30%	24%-36%	51%-69%

Table 9.8: The 95% confidence intervals for the stability of each CDC category.

A striking result from this study was that the stability values were not significantly affected by the length of time between measurements. That is, if someone is currently classified as Rejected, they have a 39-49% chance of being classified as Rejected both one month from now as well as four years from now. This independence between stability and time has important implications for the models being developed. Since the models are of individual behavior over time, the longer the model is run for, the more simulated 'time' is passing. If stability measurements are repeatedly performed, then there should be a long period of simulated time where the stability values match those in Table 9.8. That is, the model must not only match the stability data, but also continue to match over a broad range of simulated times.

9.2.3 Environment and Measures

The situation to be modelled here is considerably more complex than the repeated binary choice example. Instead of a tightly controlled experimental situation with a single participant, the unit of study is a full classroom of 30 interacting individuals, each of whom is given a questionnaire asking them to name three people that they like and three that they dislike.

To create this environment, there must be a way to generate models for thirty separate individuals. These models must be allowed to interact somehow over a period of time. Then, there must be a way to “ask” each of these models to indicate three individuals that they like and three they dislike. Clearly, this will not be through simulating the process of verbally asking the questions and interpreting the responses. Instead giving the questionnaire will be a particular input to the model, which will cause it to output references to the three models that it likes, and the three it dislikes. How these choices are made, and how they relate to the internal processes within the model, will be part of defining the particular individual models. A rough outline of this process, focused on the individual, is shown in Figure 9.7.

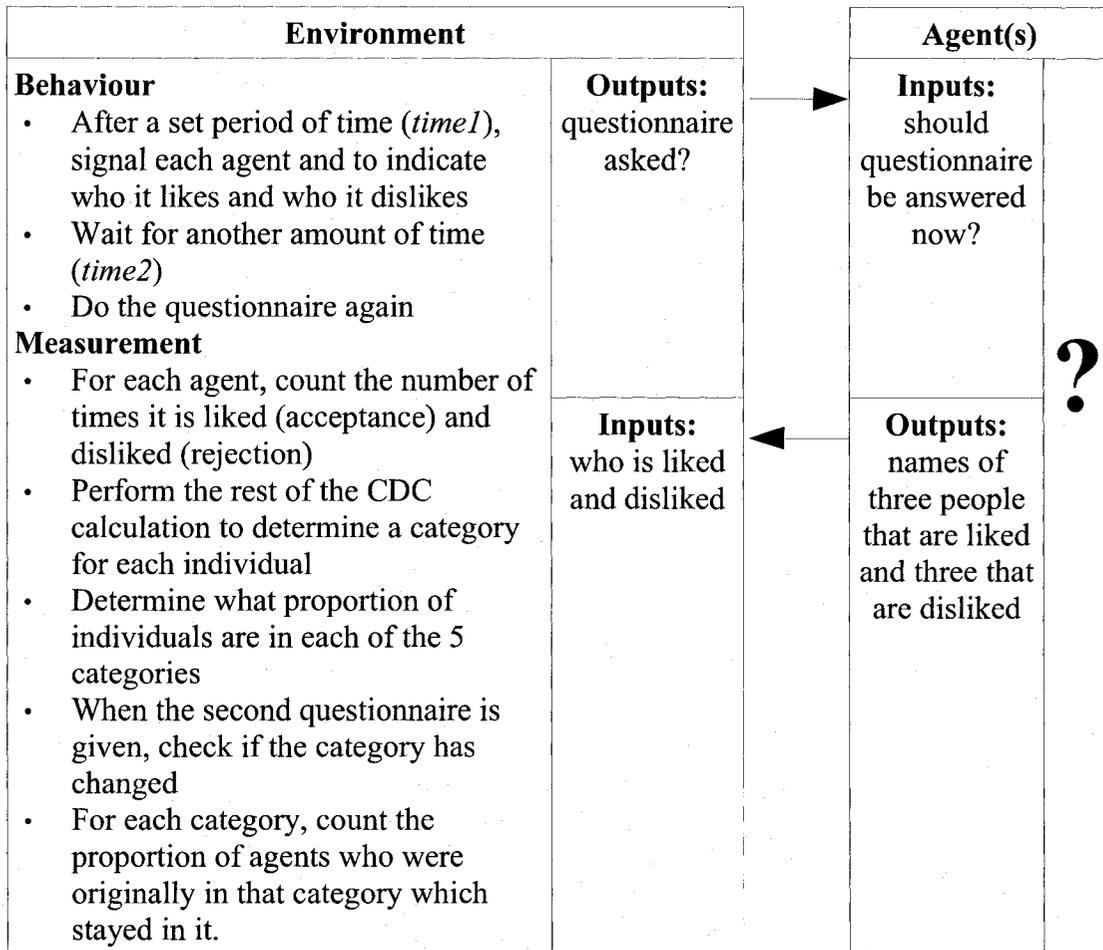


Figure 9.7: Agent-Environment interaction for the peer group environment

9.2.4 A Simple Model

The simplest possible model for this situation is one where the agents answer the questionnaire randomly. That is, they give the names of three people at random, and then three other people at random. Here, everyone has an equal chance of being chosen as liked or disliked. This represents the extreme case of the popularity categories being determined randomly, and there are no stable long-term preferences or internal state within an agent. Indeed, in this model there is no need for interaction among the individuals, as their measured behaviour does not depend on this interaction in any way.

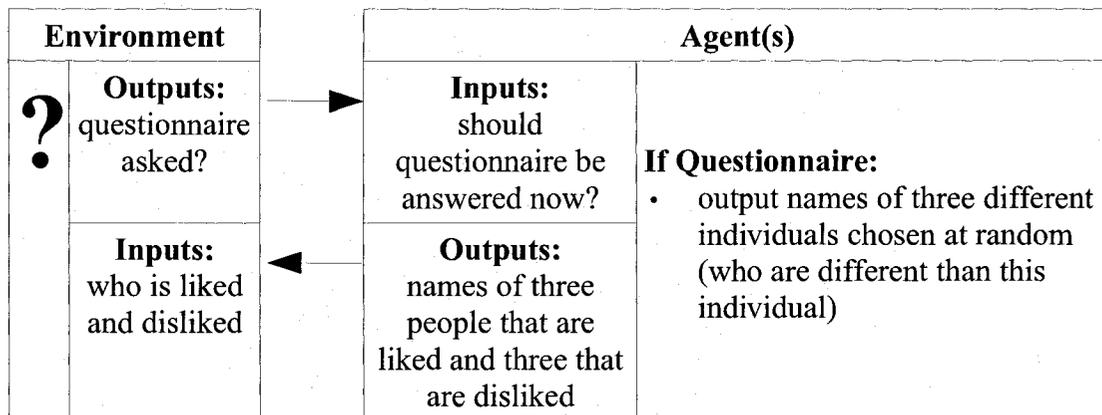


Figure 9.8: The random model for peer interactions

The results for the five basic measures (the proportion of individuals in the five categories) are shown in Figure 9.9. Surprisingly, the random model does fit within the confidence intervals of the real-world data. That is, there is no statistical evidence that this completely random model produces behaviour that is different from the real situation. This result holds no matter how long the two time parameters in the environment are (the amount of time that elapses before the first CDC measurement and the amount of time between the two CDC measurements).

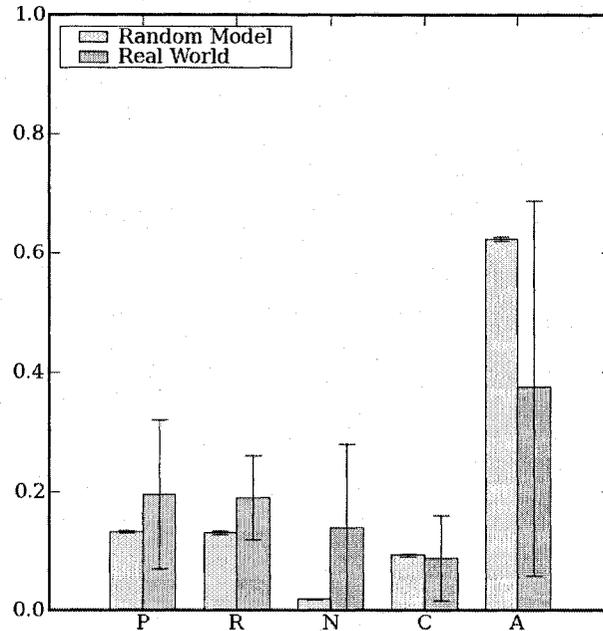


Figure 9.9: Proportion of individuals in each of the five CDC categories

This is clearly a worrying result, as the random model would be an unsatisfying explanation of popularity and rejection in peer groups. However, what this is actually saying is that *more information is needed*. One way to gain more data would be to gather evidence from more studies of this kind. This would reduce the size confidence intervals. Another method would be to add other measures to be considered. In this case, the *stability* of these categorizations could be added. It should be noted that either approach involves turning to real-world experiments, rather than further modelling work.

In this case, adding the stability measures produces the data seen in Figure 9.10. Here it is clear that the random model produces values well outside the expected ranges. The equivalence test measure here would produce values much greater than the size of the real-world confidence intervals. As a result, the random model is falsified. Later, in

sections 10.2 and 12.3, this situation will be reconsidered, and alternate models will be proposed that will produce more compatible results.

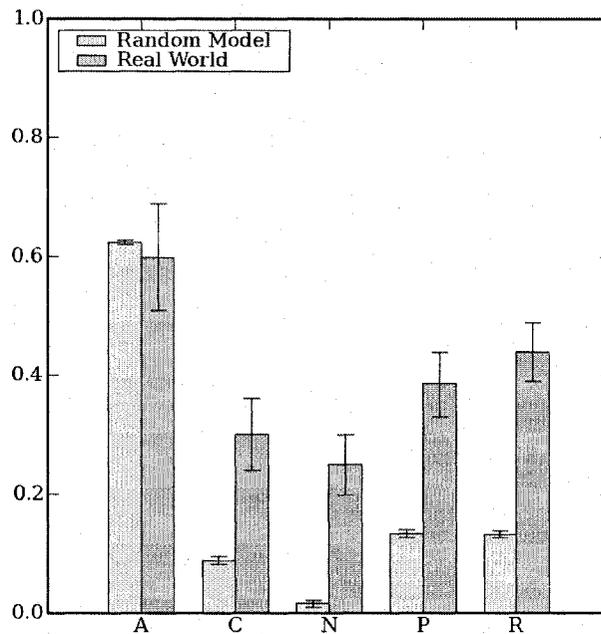


Figure 9.10: Stability of individuals in each of the five CDC categories

9.3 Summary

9.3.1 Methodology

This chapter presented two different scenarios that can occur when modelling. One requires the removal of measures, and the other requires the addition of measures.

In the first example, the RELACS model was evaluated using the measures for the 40 different conditions used in the original paper (Erev & Barron, 2005). In that paper, these measures were used to find the Mean Squared Difference between the model and reality.

This did not take into account confidence intervals, and averaged across all conditions.

Instead of taking this approach, the relativized equivalence measure was used to identify models that are equivalent to the human results across all of the measures. It was discovered that no models were equivalent on all conditions.

According to the methodology presented here, in this situation one can start removing measures from consideration in order to identify a sub-set of measures for which the model is equivalent. This should not be done by simply removing the worst measures. This leads to a situation where one cannot characterize what the measures the model work on have in common. Instead, measures were removed in related groups based on the overall group equivalence. This resulted in removing the measures for extremely simple conditions, conditions with rewards based on continuous probability distributions, and conditions where the only difference is the magnitude of the reward. One further condition also had to be removed (#40), even though it seems similar to the other conditions. To determine whether this is a real effect or a side effect of an unlucky sample, more empirical results are needed.

In the second example, a simple random model of peer groups was examined. In this case, it turned out that this extremely simple model gave equivalent performance in terms of the numbers of individuals that were categorized as Popular, Rejected, Neglected, Controversial, and Average. Since these measures are accounted for so easily, the methodology suggests that new measures be added. In this case, a measure of category stability was added, which the random model failed to account for. This indicates that new models are needed. These will be re-investigated in section 10.2.

9.3.2 Contributions

- It was found that RELACS is not a good model for all 40 of the original (Erev & Barron, 2005) situations. Even though it can achieve a Root Mean Squared Difference of 0.06 between the sample means of the human data and the sample means of the model data, there are many measures where performance is statistically distinguishable from human performance.
- However, the parameter settings $\alpha=0.01$, $B=0.05$, $L=1$, and $K=0$ (or with $\alpha=0$ and very large L) were identified as producing equivalent behaviour for conditions #2, #3, #5, #6, #8, #10, #21, #22, #23, #24, #25, #29, #30, #31, #32, #35, #36, #37, #38, and #39. These are the conditions that involve relatively simple rewards, where there are particular probabilities of having one of two different reward values for each button. RELACS is thus not a good model for situations where the rewards for the options are fixed, where the rewards follow Gaussian distributions, or where the magnitudes of the reward are the only difference.
- It was also found that RELACS also does not account for condition #40, even though it is much like the conditions it does account for. This may be due to sampling error, or it may pick out a fundamental aspect of the real situation that is not accounted for by the model. This can only be resolved by gathering more empirical evidence for this condition.

10 Model Variation

In the previous chapters, whenever changes to a model were discussed, these involved adjusting particular numerical parameters to create a new, slightly different model.

However, this is not the only way to modify a model. It is generally also possible to add or remove *components* of the model, adjusting its overall structure. Details of particular components can be changed as well. For example, if one part of a model involves taking the average of two values, this could be changed from an arithmetic average $([A+B]/2)$ to a geometric average (\sqrt{AB}) . These sorts of changes all have the potential to affect the overall performance of the model, just as any numerical parameter change.

Because of this, the effects of these non-numeric parameter changes must be explored just as the numeric parameters were in chapter 9. Of course, it will be impossible to try *every* variation, just as it was impossible to try every numerical parameter value. Instead, the modeller must explore those changes that they think may be relevant. To facilitate the task of other researchers who want to explore other possibilities, this work must, as always, be accompanied with a detailed publication of the model, including full source code.

10.1 Model Variation: RELACS

Erev and Barron (2005) also consider a number of structural variations on RELACS. For each one, they give a brief description of the change, along with their “best fit” parameter setting for the resulting model. As with the rest of their approach, this best fit is based on the Mean Squared Difference, which does not take into account the confidence interval

for the real-world data, resulting in the possibility of over-fitting. Thus, given their results, it is impossible to tell whether a particular model variation is an unsuitable model for this phenomenon, or whether it just happens to give a slightly less exact match to the particular real-world sample for the parameter values examined.

This point is worth repeating. They find that the full RELACS model, with a particular set of parameters, matches the real-world data to a Mean Squared Difference of 0.36. They then find that “RELACS with the constraint that the weighted values of the actions are updated only after selection” with a slightly different set of parameters gives an MSD of 0.40. They take this to mean that the RELACS model is a better model than the modified RELACS model, since it has a lower MSD. However, this approach to comparison does not warrant such a conclusion. As has been pointed out in chapter 4, a measurement such as MSD averages over a large collection of measures, giving no possibility for noting how much variation there is in the disagreement between the model and reality. One model could work very well for 35 of the experimental situations, and give extremely incorrect results for the other 5, while the other might work moderately well for all of the situations. This fact makes it impossible to judge which model is “better” based on only a single MSD measurement. Furthermore, since MSD only works with the *sample* mean, the quality of the match is related to the particular people used in the study, as opposed to a match to the whole population.

If the computational cognitive modelling methodology developed in this thesis is used instead, a different picture emerges. In section 9.1, the standard RELACS model was investigated. This showed that RELACS is only equivalent to human performance on

around half of the situations considered. However, this investigated only the full RELACS model (with various different parameter settings). As discussed in section 4.3.5, it is also important when modelling to examine the non-numerical parameters as well (i.e. structural changes to the model). In the case of RELACS, there are four obvious architectural changes which might be made.

Three possible changes would be removing each of the three strategies RELACS uses. The core of RELACS involves selecting one of the three cognitive strategies to use at any given moment. If one of these is removed, then the resulting model would choose from between the remaining two. Testing the resulting three models would determine if these three strategies are necessary to the performance of RELACS. This would determine if each of these strategies is a required component of RELACS.

The fourth possible change involves the system which chooses between the three strategies. Currently, RELACS uses a variant of one of the three basic strategies to choose which strategy to use at any given time. To determine if this system is necessary for RELACS to function, it can be replaced with a simpler system; for example, one which chooses randomly between the available strategies.

These four changes test whether the various parts of RELACS are, in fact, necessary. If any of them are not, they may be safely removed from the model without affecting its overall equivalence to the human data. They can be investigated in exactly the same way as the original model. It should be noted that many of these changes remove parameters from the model, so the analysis involves fewer sets of parameter settings.

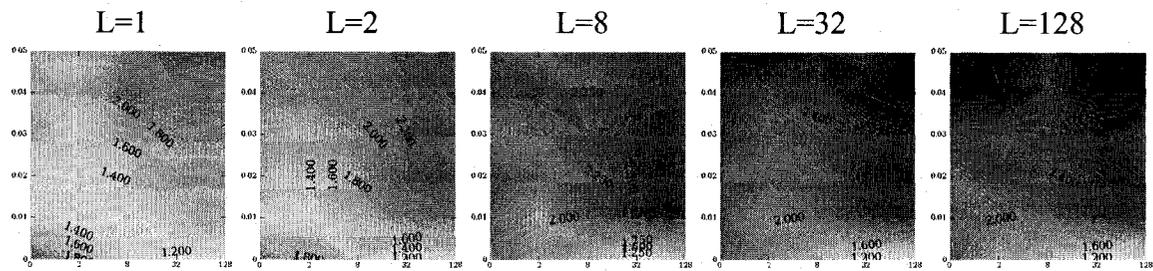


Figure 10.1: Equivalence for RELACS without the Fast Best Reply strategy. X-axis shows the K parameter and Y-axis shows the a parameter. No models are found to be equivalent to the human data.

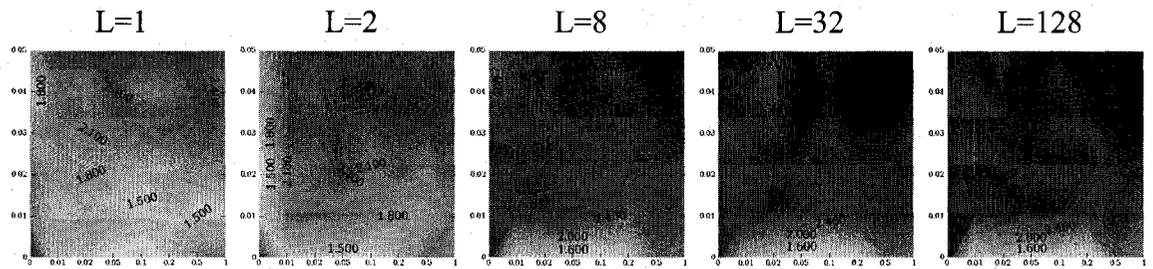


Figure 10.2: Equivalence for RELACS without the Loss Aversion strategy. X-axis shows the B parameter and Y-axis shows the a parameter. No models are found to be equivalent to the human data.

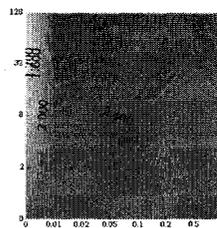


Figure 10.3: Equivalence for RELACS without the Slow Best Reply strategy. X-axis shows the B parameter and Y-axis shows the K parameter. No models are found to be equivalent to the human data.

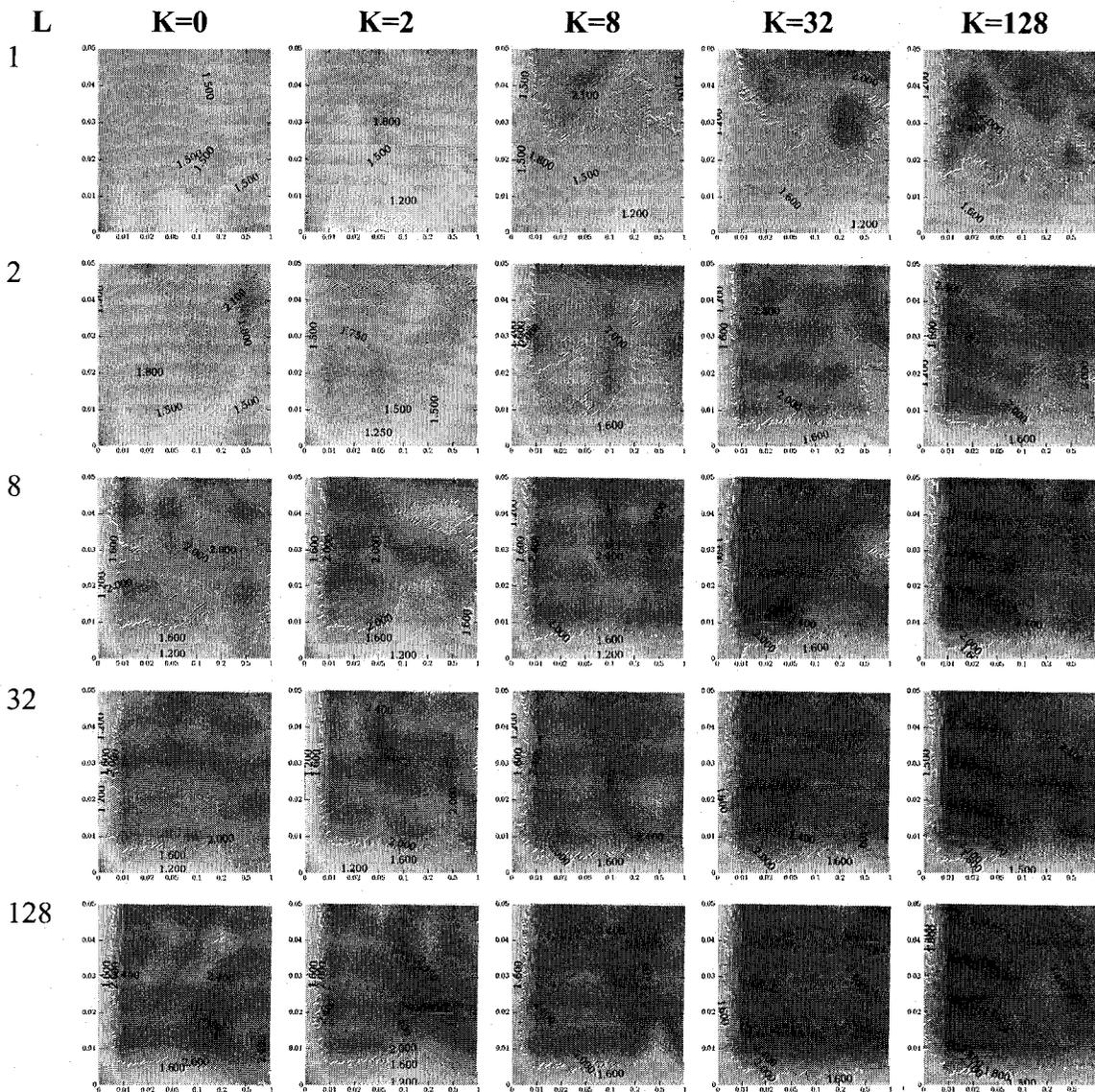


Figure 10.4: Equivalence for RELACS with random selection between strategies. In each graph, the x-axis shows the B parameter and y-axis shows the a parameter. No models are found to be equivalent to the human data.

Two main conclusions are evident from these results. The first is that removing strategies from RELACS severely reduces its performance, as all of these models seen in Figure 10.1, Figure 10.2, and Figure 10.3 were statistically different from the human results. Given this result, it is correct to conclude that all three strategies are needed. This conclusion is the same as that given by Erev and Barron (2005), but the equivalence

test measure gives a much stronger justification for this than their MSD measure.

The second conclusion can be seen from Figure 10.4. Here, the use of random selection between strategies leads to no equivalent models. This indicates that the selection mechanism used by RELACS is needed. Of course, it may be that some other system for choosing between strategies may also be equivalent, but it is at least established that something more than random selection is needed.

This conclusion is in contrast to that found in the original research. Since the original work was done using a Mean Squared Difference measure, they showed that switching from a complex selection mechanism to a random one did *not* increase the MSD measure. That is, MSD gives no reason to choose RELACS over RELACS with random selection. However, using the equivalence measure, it is clear that random selection makes the model non-equivalent to the human data.

10.2 Creating a New Model: Peer Groups

A more intensive form of model variation occurs when a new model is created from scratch. These can be seen as the extreme case of changing the structure of a model, since a whole new system is defined. As an example of this, the model of popularity and peer groups introduced in section 9.2 can be readdressed. In that section, measures of the proportion of individuals in each of 5 popularity categories (Popular, Rejected, Neglected, Controversial, and Average) were made, along with measures of the stability of these categories over time (for example, what proportion of individuals were Rejected at one point in time continued to be Rejected later). The random model was

shown to match in terms of the proportion of individuals in each category, but not in terms of the stability over time.

In these situations, when a model is falsified by the measures of interest, and there are no parameters to adjust or aspects of the model to change, new models can be created.

Later, in chapters 11 and 12, the creation of models based on existing cognitive architectures will be examined. For this section, however, a new model is created from scratch. This model, and much of the analysis found in this section has been previously published (Stewart, West, & Coplan, 2007).

Since the model is meant to address interpersonal interactions and their impact on how much one individual likes another, the following three basic characteristics were identified.

1. Each person should remember how much they like each other person
2. People should use this memory to determine how they will 'interact' with others
3. People should use the results of the interactions to change how much they like the person they just interacted with

This led to the following generic algorithm:

1. Let $a[i,j]$ be the amount person i likes person j
2. For all pairs of individuals i,j :
 - a. Use $a[i,j]$ to determine how i behaves toward j (referred to as $b[i,j]$)
 - b. Use $a[j,i]$ to determine how j behaves toward i (referred to as $b[j,i]$)
 - c. Update $a[i,j]$ based on how j behaves toward i ($b[j,i]$)
 - d. Update $a[j,i]$ based on how i behaves toward j ($b[i,j]$)

3. Repeat step 2 for as much time as is desired

It is important to note that this system models the ‘positiveness’ or ‘negativeness’ of an individual’s behaviour with a single scalar value. This means that a highly positive action is represented by a number like +10, and a slightly negative action would be something like -0.7. The same approach is taken for how much an individual likes someone else. We can thus imagine the following scenario in the model:

Two individuals (X and Y) are interacting. X likes Y a lot (+10), and Y likes X a little bit (+4). X chooses to act very nicely toward Y (+9), and Y chooses to act somewhat nicely toward X (+5). After the interaction, X and Y will change how much they like each other to new values based on the outcome of the interaction (e.g., +8 and +6).

To implement this as a model it is necessary to precisely define an algorithm for each agent to use to decide how to behave toward each other, based on how much they like each other. For this situation, a simple method is to generate a Gaussian normally distributed random value with a mean of $a[i,j]$ and a standard deviation of 1.³⁴ This value represents how ‘nicely’ individual i is going to behave toward j .

$$b[i,j] = G(a[i,j],1)$$

Similarly, an algorithm is needed to update how much i likes j , based on j ’s actions. One idea would be to simply add $b[j,i]$ (how ‘nicely’ j has behaved toward i) to $a[i,j]$.

However, this approach can be shown to cause a positive feedback loop which would mean that if i currently likes j and j currently likes i , they will keep increasing how much they like each other to larger and larger values. This clearly does not capture the ebb and

³⁴ It turns out that changing this deviation does not affect the overall behavior of this model, so it is fixed to a value of 1.

flow of real human relations. Instead, a slightly more complex formula is used which acknowledges the role that $a[i,j]$ (the amount person i likes person j) could play in determining how person i would react to the behavior of person j .

$$a[i,j] \leftarrow a[i,j] + r(b[j,i] - a[i,j])$$

The idea behind this formula is that individuals in the model evaluate the actions of others according to the expectations they have for them. This is based on the insight that one expects a friend to be friendly and an enemy to be unfriendly, and so is not surprised when this happens (and thus would not change one's beliefs). However, if a friend were to be unfriendly to the same degree, one could be quite hurt, and likewise, if an enemy were to be kind it could surprise and could cause a reevaluation of feelings toward them. This formula can be seen as a simple estimator, which means that $a[i,j]$ can be thought of as an estimate of $b[j,i]$. In other words, how much an individual in the model likes someone else is an estimate of how that other individual will behave toward them.

The long-term consequences of following even this simple model are not obvious. Since repeated iterations of this model will cause $a[i,j]$ and $a[j,i]$ to interact with one another, it is instructive to view what typically happens to these variables over time. Figure 10.5 illustrates the effect. The results indicate that while the two individuals in the simulation generally like each other by about the same amount, there is a considerable amount of change going on that could potentially capture (at least qualitatively) the ups and downs of real human relationships.

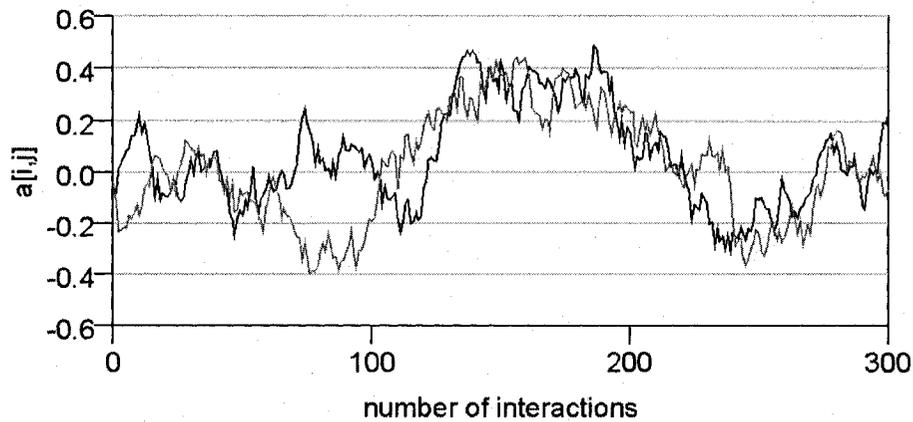


Figure 10.5: Two individuals interacting over 300 iterations of the model. Shown is $a[i,j]$ and $a[j,i]$. r is 0.05.

It should be noted that this formula introduces a parameter into this model: r . This is the *adaptation rate* and controls how quickly an agent changes its opinions of others. It can be considered to be similar to the learning rate parameters found in a wide variety of other computational models, such as neural networks (Rumelhart et al, 1986) and the Rescorla-Wagner model of Pavlovian conditioning (Rescorla & Wagner, 1972).

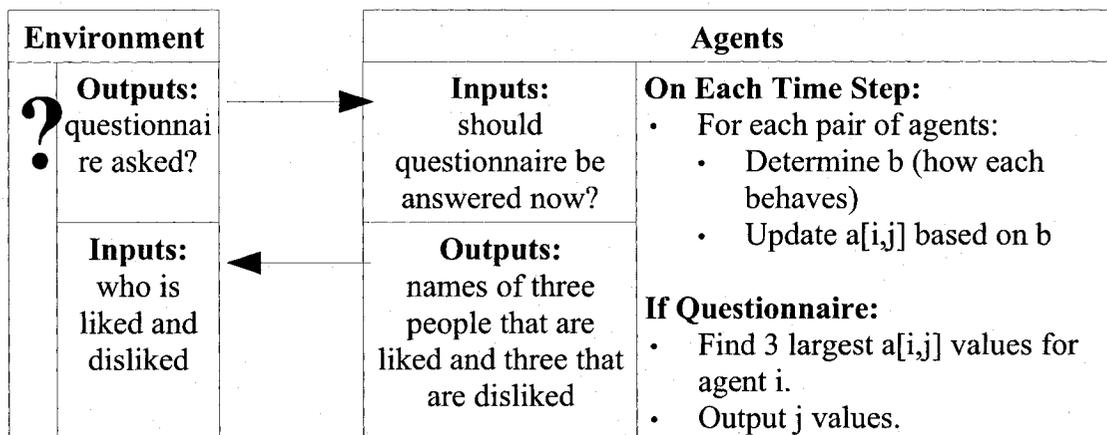


Figure 10.6: The mathematical model for peer interactions

This model can be evaluated in the same way as the random model. The model has a single parameter (r), and the environment itself has another parameter: the number of

time steps that elapse between the two CDC questionnaires.³⁵ The relativized equivalence between this model and reality is shown in Figure 10.7. The measures in question are the stabilities of the five CDC categories, which are the measures that the random model in section 9.2 was unable to account for.

The results here indicate that this model can produce outputs equivalent to the known human results. The Combined Measures graph shows a large area of the parameter space that is not statistically distinguishable from the human data (the area within the black lines). This indicates that for values of r near 0.1, and for elapsed times between 1 and 250, the model matches the real-world values. For larger periods of time, the model is only equivalent with a much higher learning rate value. Since the empirical findings suggest that these measures are not highly sensitive to the amount of time between measurements, this suggests that the parameter settings with r close to 0.1 are the most suitable for this model. This becomes a canonical parameter range for any future use of this model (see section 13.1 for an example of such use).

This model successfully captures the dynamics of peer interaction identifiable using the CDC measure. It is an improvement over the random model presented in section 9.2. For other possible models, based on the common cognitive architecture ACT-R, see section 12.3. For other implications of this model, including using it for qualitative predictions, see section 13.1.

³⁵ The environment technically also has a second parameter: the number of time steps before the first CDC questionnaire is given. This reflects the previous experience the students have had with each other, and should be large in comparison to the time between measures. For the results shown here, this value is set to 100. Reducing it increases the variability in the observed behaviour, while increasing it has no observable effect.

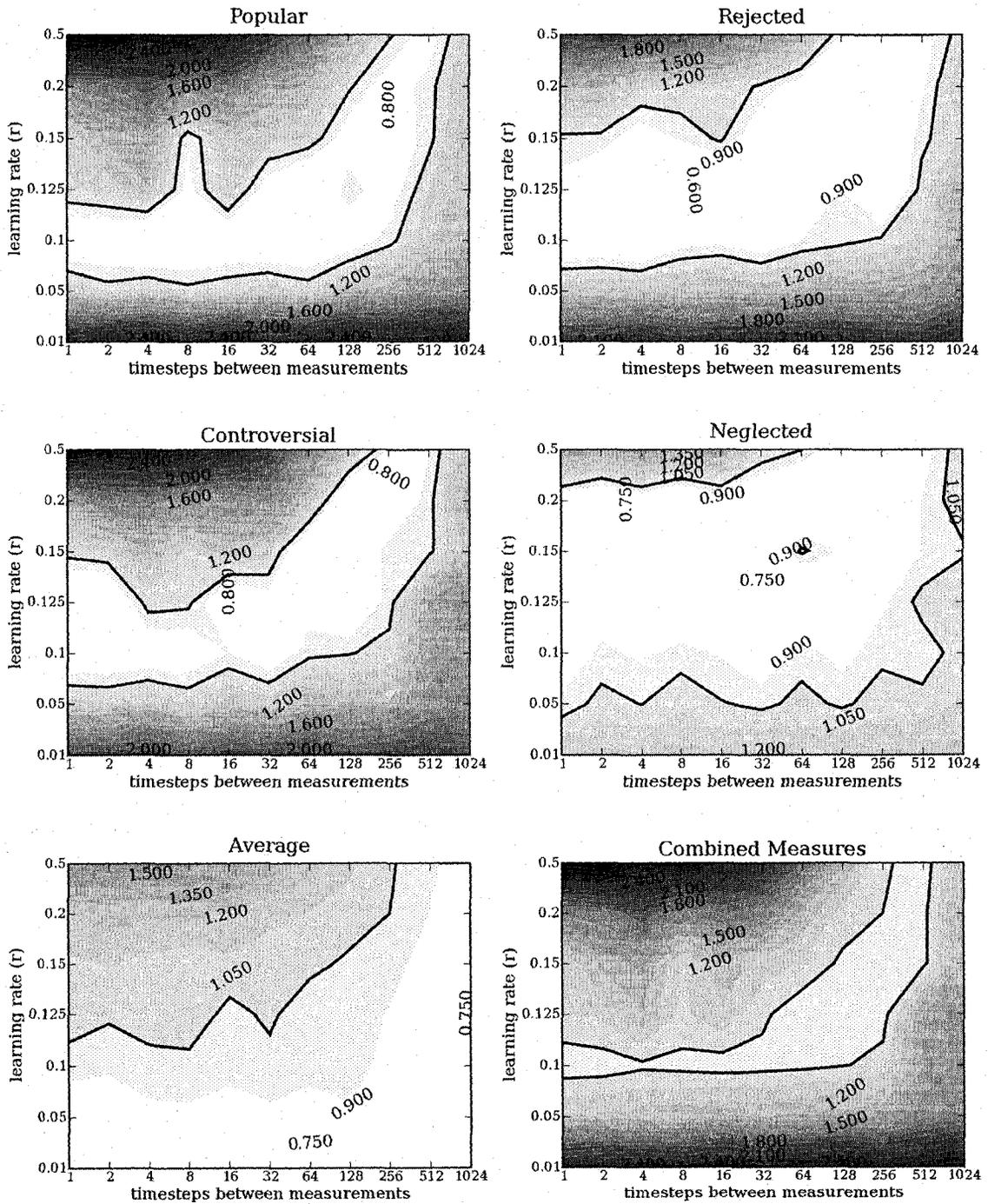


Figure 10.7: Individual and combined results for the model of peer groups. Parameter settings between the dark lines are models which are equivalent to the human data on the CDC measures.

10.3 Summary

10.3.1 Methodology

In this chapter, the main emphasis has been on investigating different models of the same phenomenon. In the first section, variations on the RELACS model were investigated. The purpose of this was to establish that the various components of RELACS were all necessary. The result was a clear indication that each of the three separate cognitive strategies in RELACS are, in fact, needed. This is a conclusion shared by the original developers of RELACS, who used an MSD measurement approach. However, when examining the system for choosing between strategies, the methodology described in this thesis leads to a different conclusion than was found in the original RELACS studies (Erev & Barron, 2005). In particular, the MSD approach indicates that using random choice makes the system no worse *on average*. However, the equivalence measure shows that using random choice leads to models which are not equivalent (although they may have the same average error). This demonstrates the importance of the equivalence-based methodology.

In the second section, an entirely new model was developed. In section 9.2, it was established that a simple random model was not equivalent in terms of the stability measures of the CDC categories (although it was equivalent in terms of the proportion of individuals in each category). As suggested by the methodology, when a model fails to account for a measure, there are two options: either remove the measure or look for new models. In this case, a new model was created from scratch, based on a simple prediction and feedback system. This new model did turn out to be equivalent to the human data

(for parameter settings of r around 0.1). However, since this model has been built specifically for this task, it is hard to relate it to other cognitive theories. In chapter 12, an alternate approach to creating new models is shown, which involves building them from an existing cognitive architecture.

10.3.2 Contributions

- It was shown that RELACS does, in fact, require the use of a high-level cognitive strategy to decide between the three underlying cognitive strategies. This is in contrast to the original results (Erev & Barron, 2005), where only the Mean Squared Difference was considered.
- A novel model of peer group interactions was developed and shown to successfully capture the long-term stability of the human peer group data as well as the basic distribution measures.

11 Modelling Architectures: Python ACT-R

In the previous chapter, a completely new model of interaction among peers was developed. This was based on original mathematical formulae, and developed specifically for that situation. While this is certainly a common occurrence in the domain of cognitive modelling, it is also common to develop models that are *based on existing cognitive architectures*. Such systems can specify a large set of cognitive components that are known to produce good models in a variety of other situations, and so are hopefully more likely to be useful in a new situation. Using such an architecture can significantly decrease the time required to create models, and significantly improve the quality of models. For example, if one were to want to modify the peer group model in such a way as to add a realistic declarative memory system (allowing individuals to forget each other, or remember certain past events more clearly than other), then a significant amount of effort would be required to create such a system, and there would be a large number of possible parameters to adjust. However, if there were a previously existing memory system that had been extensively tested in many other memory tasks, then it could be easily added to any given model.

The most comprehensively studied cognitive architecture is currently ACT-R (Anderson & Lebiere, 1998). This system consists of a declarative memory module, a procedural memory module, a perceptual/motor system, and a variety of rules for how these systems act over time. This has been used extensively by a wide variety of researchers for such domains as sentence parsing, the Stroop test, spatial reasoning, skill acquisition, scientific discovery, and so on.

Unfortunately, the current software implementation of ACT-R is not designed to allow for easy manipulation *of the architecture itself*. That is, modifications are easy to make to the contents of declarative and procedural memory, but it is much more difficult to combine ACT-R components with other new aspects of models. This, for example, makes it less practical to simply add the ACT-R declarative memory system to the peer group model described above. Furthermore, while certain parameters can be adjusted within the ACT-R system, it is much more difficult to adjust whole aspects of how the various modules function. In order to adjust these aspects of the model, extensive knowledge is required of the underlying Lisp implementation of the model.

In order to allow ACT-R to be easily integrated with other models, used in various different situations, and to allow for further exploration of different variations of the ACT-R components, a new version of ACT-R was developed. This was based on the published formulas and behaviour of the official Lisp ACT-R, but was integrated with the CCMSuite system, making it easy to mix and match components with other models, to adjust components, to develop entirely new systems, and to then have those systems interact with the environmental situations defined using CCMSuite. The goal is thus to allow for the exploration of the architectural space, in terms of comparing different ways that components of the ACT-R architecture could be implemented. That is, it supports exploring ideas that might contribute to the next, official version of ACT-R. This chapter is adapted from the main publication describing this process (Stewart & West, 2007).

One way of creating such a system would have been to delve into the Lisp version of ACT-R 6. However, it was decided to re-implement ACT-R 6 in the Python

programming language instead (which is the basis of CCMSuite). One reason for doing this is that re-implementing ACT-R based on the theory, and not the Lisp code, demonstrates (to skeptics) that ACT-R is a *theory* and not a piece of software full of tricks for fitting human data (a belief frequently encountered outside of the modelling community). A second reason was that the process of rewriting from scratch helped to deconstruct ACT-R into more fundamental components.

11.1 Deconstructing ACT-R

This deconstruction of ACT-R began by noting that ACT-R is constructed from three basic components: a chunk-based communication system (chunks and buffers), a chunk storage system (the declarative memory system), and a pattern-matching production system (the procedural memory system). ACT-R provides a theory of how these components are combined, and a theory of how each works. These are described below, followed by a description of how Python ACT-R builds on this to create a system for the specific purpose of exploring the architectural space.

However, before continuing, it should be noted that two high-level decisions shaped the re-implementation. The first is that the syntax for writing ACT-R models was modified. This was originally done to ease the integration of ACT-R into the Python programming language. However, as a byproduct, it also made the distinction between the core ACT-R theory and the particular syntax used to write models more clear. The result is a system that is functionally equivalent to Lisp ACT-R, but does not make the same syntax choices. It was also decided that Python ACT-R would be as explicit as possible in terms

of what is happening within the model. In Lisp ACT-R, there are many side-effects; situations where code in the model that explicitly does one thing also causes other actions to be performed that are not explicitly represented in the model code. This leads to programming efficiency and makes certain parts of the theory automatic. By eliminating side effects and requiring the modeller to explicitly represent them in the model, it is easier to modify these effects.

11.2 Communications

ACT-R is a modular theory of mind. That is, it treats the mind as being composed of distinct modules that exist for particular functions. This being the case, the modules need to communicate to each other using a common mechanism.

11.2.1 Chunks

In ACT-R the various components of the architecture communicate using a simple symbolic representation system, called a chunk. Each chunk has a number of slots, each of which contains a single symbol. These symbols can represent anything (including other chunks), but do not have inherent semantic value. As a guideline, it is recommended that chunks have a small number of slots. Miller's number of 7 ± 2 is recommended as an upper limit (Anderson & Lebiere, 1998), although this is not enforced.

To do anything with the slots of chunks, there must be a way to distinguish between them. In Lisp ACT-R, this is done by giving each slot a name. Python ACT-R allows for this, but can also distinguish slots by position instead. For example, the following text

shows how a chunk representing a large, friendly dog might be represented in Lisp ACT-R and Python ACT-R. We will use the Python ACT-R syntax throughout this paper, and will not make use of slot names both for simplicity and to underscore the fact that there is no semantic meaning attached to particular slots.

```
Lisp:                (chunk isa dog size large manner friendly)
Python with slots:   chunk='isa:dog size:large manner:friendly'
Python without slots: chunk='dog large friendly'
```

In Lisp ACT-R, one particular slot has been treated differently from the other slots. The isa slot (the first one in the above example) is meant to indicate what type of chunk it is. Chunk matching (as described later) to the isa slot used to be necessary in Lisp ACT-R, but this is no longer required as of ACT-R 6. This change suggests that the special treatment of the isa slot is no longer a core component of the ACT-R theory, so we have not included it in Python ACT-R as a computationally enforced modelling constraint.

11.2.2 Buffers

The other component used for communication between modules is the buffer system.

Buffers are capable of holding only one chunk at a time. Modules can place a chunk into a buffer, modify the value of slots of a chunk, or clear the buffer. They can also retrieve a copy of a chunk in a buffer. All of these actions are performed instantly. Importantly, the chunk it contains is a copy of the original chunk, meaning that changes to the contents of the buffer do not alter original chunk.

Taken together, the buffers create a representation of what might be called context. That is, because they hold the most recent output of each module, they more or less

collectively represent the current state of the whole system. This represents a strong claim of the ACT-R theory, that the state of the system is represented by a limited number of chunks in the buffers. Also, the buffers are considered to be physically separate from other ACT-R modules (Anderson et al, 2003). This means that the buffers represent particular areas in the brain.

11.2.3 Chunk Matching

One of the fundamental mechanisms for working with chunks is to be able to find matches. There are two ways this is used: first, to examine the current context (as defined by the contents of the buffers) to determine what action to take next (i.e., what production to fire), and second to search for a particular chunk among a large collection of chunks (such as might be stored in declarative memory). The details of such usage are described later, in the sections about the appropriate modules. However, in all cases, the matching process involves determining whether or not a particular chunk matches to a particular pattern. A pattern is a chunk whose slots contain matching rules, rather than actual contents. A pattern matches with a given chunk if each of the slots in the pattern has a rule that matches with the contents of the corresponding slot in the chunk.

There are three matching rules.³⁶ The first is an exact match, where the pattern slot indicates the exact content that must be in the chunk slot. Second is a 'not' match, where the pattern indicates a value that the slot cannot have. In the Python syntax we use a ! to indicate the 'not' match, so the pattern 'dog !small friendly' would match with

³⁶ There are modules that have other sorts of matches, such as numerical comparisons. These module-specific additions are not necessary for implementing the core theory of ACT-R.

the chunk 'dog large friendly' but not 'dog small friendly' or 'dog large vicious'. The third matching rule matches to the situation of having no chunk at all. This handles the special case of matching to an empty buffer.

Also important for the matching process is the idea of a variable. This is an arbitrary label that can take on different values when it is placed in the slot of a pattern. The first time a variable is used within a match, its value is bound to the value in the corresponding slot. In Python ACT-R, we use a ? to indicate variables. For example, given the chunk 'dog large friendly' and the pattern 'dog ?size friendly', the variable ?size would be bound to the value large. Once a variable is bound, further uses of that variable within that context must have the same value.

Thus, the pattern 'cat ?size sleepy' would now match to the chunk 'cat large sleepy' but not 'cat small sleepy' because the variable ?size has been bound to large. One can also use an ! to indicate a not-match for variables (such as 'cat !?size sleepy').

Since there are multiple buffers in an ACT-R model, patterns can be specified across them. The bound variables must have the same value across all of the buffers. For example, if there are two buffers (buffer1 and buffer2) and the matching pattern was:

```
buffer1='dog ?size friendly',buffer2='cat ?size sleepy'
```

it would match to either of the following buffer contents:

```
buffer1='dog large friendly',buffer2='cat large sleepy'  
buffer2='dog small friendly',buffer2='cat small sleepy'
```

but would not match in this case:

```
buffer1='dog large friendly',buffer2='cat small sleepy'
```

The variables and matching rules can be combined in any manner, including having multiple rules for the same slot (in which case all rules must match; the main use for this would be multiple 'not' matches, e.g., to match to an animal that is not a dog and not a cat).

There is also an optional system currently used only by the ACT-R declarative memory system (described below) that allows *partial* matching, where the modeller can explicitly specify that two different slot values can be considered to be “close enough” for a match. For example, the slot value `pink` could be considered to be somewhat close to the slot value `red`. The modeller can also specify a numerical “penalty” for this sort of matching, on a case-by-case basis. In the case of declarative memory, this value is subtracted from the activation level (described later) of the chunk. However, there is no overall theory specified by ACT-R as to how these penalty values should be defined.

11.3 Production System

Modules in ACT-R represent modules in the brain. Therefore, like the buffers, there is an expectation that they correspond to brain areas. ACT-R theory, therefore, divides the brain in two types of systems: functional systems (modules) and communication systems (buffers). The small chunk size recommended for the buffers can be seen as representing limited channels of communication between the modules. Modules are implemented such that when conditions require the use of a module it is activated instantaneously, in

simulated time (obviously this requires some amount of computational time but it is not added to the simulated time). Once activated, modules calculate the total cost in time for the action they are going to do. Once this amount of time has passed from when they were activated the action is instantaneously (in simulated time) carried out. All modules and buffers operate in parallel.

The ACT-R production system is a module that contains a collection of *if-then* rules for accomplishing tasks and coordinating cognition, perception and motor actions. The production system's job is to determine what production to fire (i.e. what action to carry out) at any given moment. Firing takes 50 milliseconds of time, and only one production can fire at a time. If no productions can fire, then the system does nothing. The productions are initially hand-designed by the modeller based on their theory as to how a particular task is performed, but the system can generate new productions through the production compilation mechanism.

The *if* part of a production (referred to as the Left-Hand Side in the ACT-R literature) is a collection of matching patterns, as described above. Whenever these patterns match, the production can fire. Production patterns may use all of the buffers in the model or only a subset. Overall, this forms a constraint that any information being used for decision-making by the production system *must be present in one of the buffers in the system*.

In addition to this, the *if* conditions can also include matches to the state of a module. To accomplish this, a module can include a separate buffer that holds information about its state. The particular slots and contents for this state buffer are specific to each module,

but would include information such as whether the module is currently performing an action.

The *then* part of the rule (the Right-Hand Side) consists of a series of actions to be taken when the rule fires. The actions are commands for the other modules or buffers. In the case of buffers, commands can include setting or changing the values of chunks within the buffers (or emptying the buffer). For modules, the commands (referred to as requests) can trigger modules to perform any action that a module can do. ACT-R theory implies that there are restrictions as to how many different sorts of actions may be performed on the right-hand side (e.g. one cannot make more than one request to declarative memory at a time).

Any bound variable from the *if* part can be used in the *then* part of the production, but after the production is fired the bound variable is discarded. This means that the power of using variables is limited in time and in space (i.e., variables created within the production system module cannot be used outside of it).

In the implementation of this process, the production system instantaneously finds all the matching productions. This may require actual computation time, but it does not affect the elapsed simulation time (i.e. no other actions are happening while the search for matching productions occurs). If a match is found, then the production system waits for 50 milliseconds. During this 50 milliseconds, the other modules may perform various actions (such as retrieving chunks from declarative memory, or moving attention in the visual system). After this, all the commands on the *then* side of the production are

executed. This is also treated as an instantaneous action. However, the requests generally will not be immediately carried out as each module calculates the time delay for the actions. Once the requests are made, the production system searches again for new productions to fire.

If no productions match, then the production system does nothing. However, the moment a buffer's content changes such that a production could fire, the production system notices this and readies that production to fire 50 milliseconds later. The production system must thus perform its search for matching patterns any time there is a change to the buffers and it is not currently waiting to fire a previously matching production.

The production system must also guarantee that the contents of the buffer are still a valid match after the 50 millisecond delay. That is, even if the contents of the buffers have changed since the beginning of the 50 millisecond delay, the changes must not be ones that affect the matching, otherwise the production could be inappropriate. In Lisp ACT-R it is unclear to us whether the pattern must match at all times within the 50 millisecond time frame, or just at the beginning and end. Also, it is unclear if the production system matching restarts immediately upon a buffer change, or if it waits until the 50 milliseconds are over. If this change occurs in Python ACT-R the production selection process is restarted after the 50 milliseconds without firing the previously selected production.

Another issue is that in Lisp ACT-R requests are sent to modules by placing the requests in a buffer associated with the module, whereas in Python ACT-R the requests are sent

directly to the module. Using the buffer in Lisp ACT-R does not impose any time delay, so the only constraint it imposes is that the request should be a chunk, and thus should be of limited size. This constraint is no different for Python ACT-R, and is not enforced in either of them.

11.3.1 Production Conflict Resolution

The core decision-making aspect of an ACT-R model is how it determines what to do when multiple productions match. Since only one can fire at a time, there needs to be a method for choosing which one. In ACT-R, this is done by estimating a production's *utility*. Whichever matching production has the highest utility is the one that will fire.

The standard approach for calculating utility (U) is based on the following formula:

$$U_i = P_i G - C_i + \varepsilon$$

The three parameters are: the probability that the production will lead to a success (P), the value of that outcome (G), and the amount of time that will occur between performing this production and achieving that outcome (C). Both P and C are specific to a given production, and G is an overall parameter for all productions. ε is an optional random noise value with adjustable variance, usually set to 0.

It is possible to simply set each of these values manually, but it is desirable to create models that will learn values for these parameters on their own. If this mechanism is used, then there must be some way to specify successes and failures. In ACT-R, this is done by marking certain productions as indicating success, and others as indicating

failure. The system keeps track of the number of times that firing each production eventually leads to a success (s), and how many times it leads to a failure (f). It also keeps track of the total amount of time between a production firing and either a success or a failure occurring (t). Given this information, P and C are estimated as follows:

$$C = \frac{t}{s + f} \qquad P = \frac{s}{s + f}$$

There are also alternate versions of these formulae, which include the ability to record multiple successes and failures at once (Gray, Schoelles, and Sims, 2005). Since this is a developing area of research, Python ACT-R includes these variations, and makes it easy to create and modify mechanisms for adjusting the utility of productions.

ACT-R also defines a system for the generation of new productions out of old ones, known as *production compilation*. Here, two productions are combined into a single production that performs both actions. Importantly, this new production must be constrained to only match in situations where the two original productions would have fired one after the other. This mechanism is complex enough to take into account situations where the second production fires as a result of a declarative memory retrieval (as will be discussed in the next section). That is, instead of one production requesting memory recall, and the second production making use of that information, the result is a single production which performs the special-case task of executing the correct action in the situation, without requiring explicit memory recall. This is meant to model the transition from having to explicitly recall what to do next to simply automatically

performing that action.

11.4 *Chunk Storage*

The declarative memory system in ACT-R is a module for storing and retrieving chunks. There is no limit on its capacity, other than its mechanism for making chunks harder to retrieve the less often they are used. It is generally thought of as a storage system for explicitly known information.

Adding a chunk immediately places it into whatever internal storage mechanism is being used in the given implementation. If two identical chunks (i.e. ones with all slots with identical values) are added, then the system treats them as the same chunk. This mechanism is called merging and is used to strengthen the activation of the chunks, as will be described below.

In Lisp ACT-R, chunks are normally added at the very beginning of the model (representing background knowledge). There are then various rules indicating when the chunks in various buffers should be automatically added to declarative memory. As discussed above, in Python ACT-R the modeller needs to explicitly state when chunks should be added. That is, in Python ACT-R it is the modeller rather than the software that implements this aspect of ACT-R theory. This makes it easier to write “illegal” code, but it also makes it easier to experiment with this part of the ACT-R theory.

When requesting a retrieval from memory, a *pattern* (as described above) is given to the module. The declarative memory system will then find the chunk that matches the pattern and has the highest activation value, and place it into a particular buffer known as

the retrieval buffer. The amount of simulation time taken is based on the following formula:

$$t = Fe^{-A}$$

Here, F is a parameter called the latency factor, which is set by the modeller. A is the activation of the chunk. If no chunks have an activation higher than the threshold (which is also set by the modeller), then no chunk is placed in the buffer.³⁷ The amount of time this takes is determined by substituting the threshold into the above equation instead of A .

There are a variety of mechanisms in ACT-R for determining the activation of the chunks. They can be set manually, but this practice is very rare. Almost always they have a certain amount of random noise (ϵ) associated with them. The main method for adjusting the activation is known as base-level learning. Here, the activation is adjusted based on the use of the chunk. When a chunk is used (i.e. whenever an identical chunk is added), this increases its activation. Conversely, the activation level gradually decreases when it is not used. Exactly how this process is best modelled is a subject of much study, resulting in a number of different approaches:

$$A = \ln(\sum t_i^{-d}) + \epsilon$$

$$A = \ln\left(\frac{n}{1-d}\right) - d \ln(L) + \epsilon$$

$$A = \ln(\sum t_i^{-d_i}) + \epsilon, d_i = c e^{-A} + a$$

³⁷ If this occurs, the module is said to be in an error state, which is one of the slots in its implicit state buffer (along with whether or not it is busy), as described earlier. This error slot will continue to be true until the next retrieval request is made.

The first formula is based on extensive experimental results (summarized in Anderson & Lebiere, 1998). t_i represents the times in the past that this chunk has been added (measured in seconds before now). The parameter d controls how quickly the activation decays, and is always set to 0.5 to match known human data. The second formula is an first-order approximation of the first, where n is the number of times in the past the chunk has been added, and L is the amount of time since the chunk was first created. This formula tends to be used for reasons of computational efficiency, but recent results (Sims & Gray, 2004) have shown that it gives significantly different results from the first equation in situations where the chunk is encountered on an irregular or clustered basis, and so this optimized version is not recommended.

The third formula is a more recent system created by Pavlik and Anderson (2005) that allows d to vary based on the current activation of the chunk. Using a chunk when it has a high activation leads to faster decay, while using it when it has low activation gives less decay. Instead of specifying d , this requires specifying two parameters, c and a . The resulting behaviour more closely matches spacing effect phenomena in memory studies (Pavlik & Anderson, 2005). All of these are available in Python ACT-R.

12 Using Python ACT-R

The previous chapter defined the ACT-R architecture and discussed its integration with CCMSuite, the system developed for creating computational models as part of this thesis. The main point of this is to allow for the easy creation of models which make use of the ACT-R theory. Thus, instead of creating a new model from the ground up (as was seen in section 10.2, which presented a novel model of peer group interaction), it is instead possible to create models from the basic components provided by Python ACT-R. In this chapter, ACT-R models for the repeated binary choice task and the peer group interaction situations are discussed.

12.1 ACT-R and Repeated Binary Choice

Given the ACT-R system as described in chapter 11, it is remarkably easy to create an ACT-R model for the repeated binary choice task. The most direct way is to use the procedural memory system to learn to press A or B. In this situation, there are two competing productions: one for pressing A and one for pressing B. Initially, they both have equal utility, so the production system will choose between them randomly. When rewards are received from the environment, these can be treated as rewards for the production system. This allows it to use the PG-C learning rule (see chapter 11) to adjust the utilities of the two productions. This, in turn, will affect how likely the options are to be chosen.

The high-level diagram for this model is shown in Figure 12.1, and the full commented source code for this model can be seen in Figure 12.2.

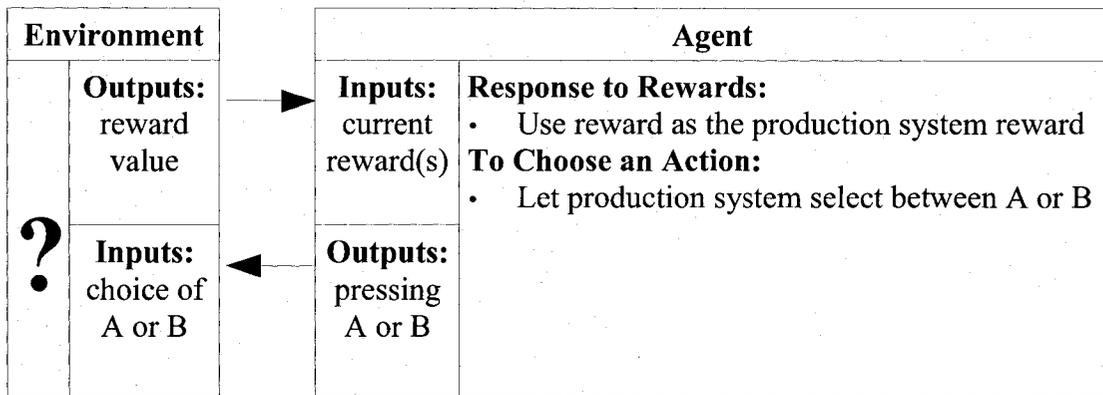


Figure 12.1: Overview of the procedural memory-based ACT-R models of repeated binary choice.

<pre> class ACTRPMModel (ACTR) : pmpgc=PMPGC () pmnoise=PMNoise (noise=0.3) def pressA (top='ready:True') : self.parent.pressA () self.reward (self.parent.reward) def pressB (top='ready:True') : self.parent.pressB () self.reward (self.parent.reward) </pre>	<p>create an ACT-R model use the PG-C production utility learning rule, with 0.3 noise</p> <p>this production presses A and uses the reward as feedback</p> <p>this production presses B and uses the reward as feedback</p>
--	--

Figure 12.2: Source code for an ACT-R model of repeated binary choice using the PG-C production system.

There are two parameters for this model: a numerical one and a non-numerical one. The numerical parameter is the amount of noise added to the production utility values. A larger noise value would cause there to be more of a random choice between the two options. While 0.3 is a common standard value for this, a wide range of values has been used in different ACT-R models. This means there is no good canonical value to choose, so many different values should be explored.

The non-numerical parameter is which production utility learning rule to use. The standard ACT-R rule has been the PG-C one, but various modifications have been

suggested (e.g., Gray, Schoelles, & Sims, 2005). This includes the Success Weighted rule, the Mixed Weighted rule, and the New Utility rule. Each of these models can be evaluated (with different noise levels) for equivalence with the human data. The idea here is to compare it to the RELACS model to determine whether RELACS captures aspects of the cognitive phenomenon that are not exhibited by these ACT-R procedural memory models.

12.1.1 Results

As in section 9.1.1, the experimental conditions for evaluating the model are divided into three groups. The first group, which examines the payoff variability effect, consists of the eight conditions successfully modelled by RELACS. None of the ACT-R procedural memory systems were equivalent on all eight measures. To attain equivalence, conditions #2, #8, and #21 had to be removed. There is no apparent pattern behind these particular conditions, however. That is, unlike the case seen in section 9.1.1, where the conditions that were removed were of a particular kind (e.g. fixed rewards), here the problematic measures are for conditions that are much like the successful conditions. There is thus no high-level way to express what measures this model is equivalent to, other than merely listing the conditions themselves. This is an unsatisfactory way of removing measures, and does not afford conclusions about the capabilities of the model.

A similar situation is evident for the other two categories of measures. The underweighting category required the removal of one out of three conditions. The loss rate effect category was considerably worse, requiring six out of the eight measures to be

removed. Again, there was no systematic pattern evident in terms of which measures needed to be removed. These results are shown in Figure 12.3. It should be noted that there are no parameter settings which overlap between the three displayed graphs, indicating that even after 10 of the 19 measures are removed, there are still no models which are equivalent on all of the remaining 10 measures.

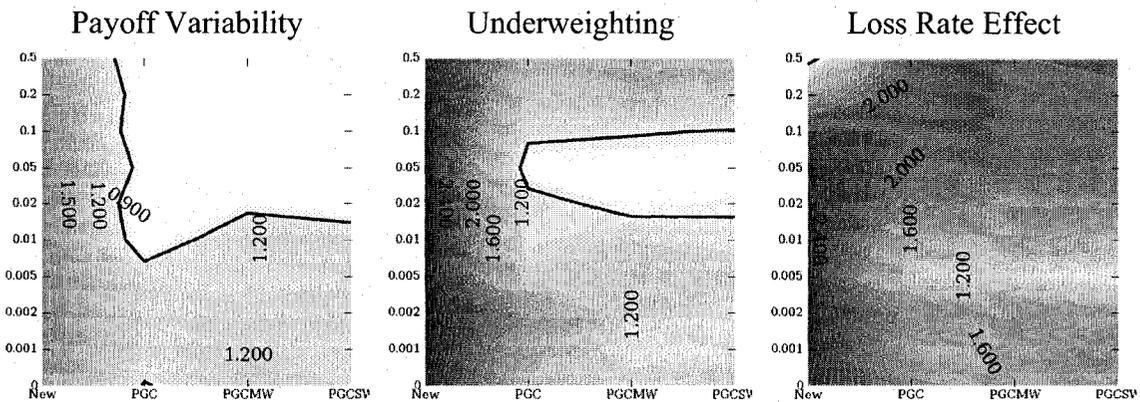


Figure 12.3: Relativized equivalence of the procedural memory ACT-R model for the three categories of repeated binary choice measures. X-axis indicates the four different learning rules, and the Y-axis shows the level of noise.

Given this result, it is safe to conclude that the basic ACT-R model for this task does not capture these aspects of the known empirical data. RELACS is a considerably better model (i.e. it is equivalent across more conditions) than any of these ACT-R based models.

#	Reward A	Reward B	
2	11	50%: 19; 50%: 1	(removed)
3	50%: 21; 50%: 1	10	
5	-10	50%: -21; 50%: -1	
6	50%: -19; 50%: -1	-11	
8	50%: 21; 50%: 1	10	(removed)
10	-10	50%: -21; 50%: -1	
21	80%: 4; 20%: 0	3	(removed)
22	20%: 4; 80%: 0	25%: 3; 0%: 0	
23	10%: 32; 90%: 0	3	
24	2.5%: 32; 97.5%: 0	25%: 3; 75%: 0	(removed)
25	-3	10%: -32; 90%: 0	
31	70%: 6; 30%: 2	30%: 6; 70%: 2	(removed)
32	70%: 4; 30%: 0	30%: 4; 70%: 0	(removed)
35	70%: -2; 30%: -6	30%: -2; 70%: -6	(removed)
36	63%: 6; 27%: 2; 10%: 0	27%: 6; 63%: 2; 10%: 0	
37	63%: -2; 27%: -6; 10%: 0	27%: -2; 63%: -6; 10%: 0	(removed)
38	63%: 6; 27%: 2; 10%: 0	27%: 6; 63%: 2; 10%: 0	(removed)
39	63%: -2; 27%: -6; 10%: 0	27%: -2; 63%: -6; 10%: 0	(removed)
40	-3	80%: -4; 20%: 0	

Table 12.1: Experimental conditions that had to be removed to find the equivalent models shown in Figure 12.3. Even with all of these removed, there are no ACT-R procedural memory based models which are equivalent on all of the remaining measures.

12.2 ACT-R, Sequential Dependencies, and Repeated Binary Choice

There is also an alternative method for using ACT-R to model a task such as the repeated binary choice task. As discussed in chapter 11, ACT-R has two major learning systems: procedural memory and declarative memory. In the previous section, procedural memory was used to represent the choices between actions. It should be noted that there was no complex processing involved in this model: the procedural memory system simply chose which one of two actions to take. This makes the previous model one in which no explicit memory recall is occurring; instead, actions are chosen 'instinctively'. While this may be appropriate for a model of an *expert* at repeated binary choice, or a model of someone who is performing repeated binary choice while being distracted by another

task, this may not be the best model for the situation.

In contrast to the procedural memory approach, an ACT-R model can use declarative memory to store *sequential dependency* information about the task. The approach, developed by Lebiere & West (1999) has been used to model implicit learning and game playing (Lebiere & Wallach, 2001; Lebiere & West, 1999; West, Stewart, Lebiere, & Chandrasekharan, 2005). The core idea is to record patterns of events over time, and then use this memory to predict what will happen in the current situation. The final choice of what action to take can be made based on this prediction. This appears to be a pervasive aspect of human pattern identification, and so is a likely candidate for modelling any sort of repetition-based situation.

For example, in the ACT-R sequential dependency model of playing Rock-Paper-Scissors (West et al., 2005), the declarative memory is used to store sequences of opponent's actions. If the opponent played Rock, then Paper, then Rock again, the sequence "Rock Paper Rock" would be stored in memory as a chunk. The more times this sequence is seen, the higher the *activation* for that chunk of memory, as per the equations shown in chapter 11. Given this memory, the current actions of the opponent can be used for prediction. If the opponent's last two plays have been Rock and then Paper, a memory search is made for chunks that match the pattern "Rock Paper ?". Whatever chunk is retrieved is treated as a prediction for what the opponent will play next.

Adapting this model for the repeated binary choice task is relatively straightforward. In

this case, the sequential actions are the choices made as to which button to press.

However, in addition to this, each chunk also contains a value indicating whether the reward associated with the final action is higher or lower than average. The resulting chunk may look like "A B B above", which indicates that pressing B after pressing A and then B led to a result that was above average.

This memory can be used in much the same way as the Rock-Paper-Scissors model. If the previous two actions were A followed by B, a search is made for a chunk matching "A B ? ?". If none are found (as will be the case at the beginning of the trial), the model chooses randomly. If it recalls a chunk that gave an above-average reward, it chooses to perform that same action again. If it recalls a chunk that was below average, then the opposite action is chosen.

A diagram of this model is given in Figure 12.4. There are three parameters for this model. First is the *lag*, which indicates how long the sequential dependencies are. The example shown above uses a lag of two, since two previous actions are used to choose the next one. In other sequential dependency models, this parameter generally best fits human data when set to two, and generally performs very poorly above three³⁸. The second parameter is the declarative memory noise, which affects how variable the performance is (i.e. how often a memory with lower activation gets recalled). The final parameter is a structural (non-numeric) parameter, which indicates whether the remembered history is a list of actions (e.g. "A B B") or a list of rewards (e.g. "11 1

³⁸ Using a lag above 3 means the system will learn very slowly, as there are many more combinations of possible events to encounter.

19"). This allows more flexibility in the response.

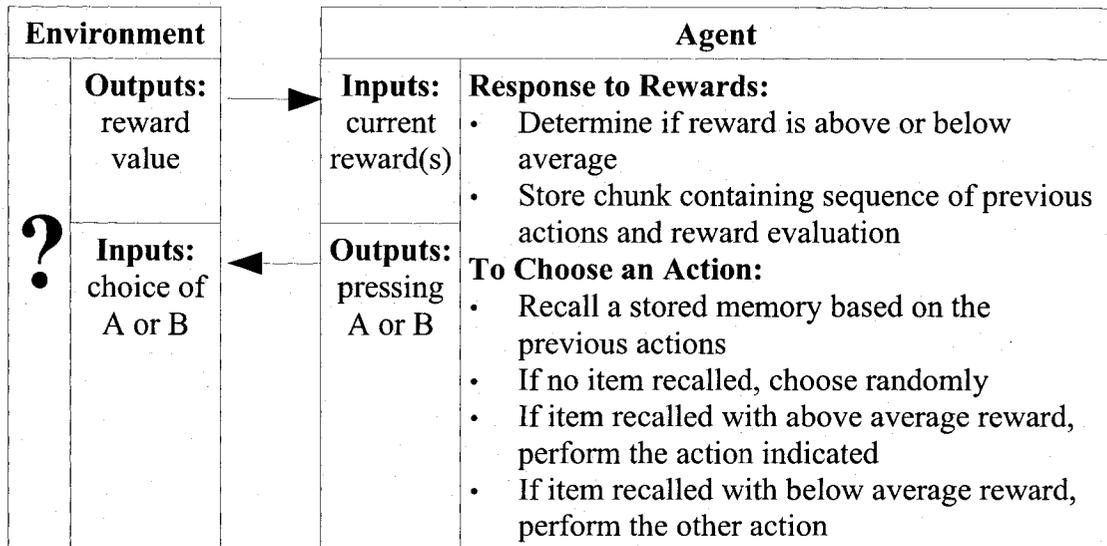


Figure 12.4: Overview of the sequential dependency and declarative memory ACT-R model of repeated binary choice.

12.2.1 Results

The results of the declarative memory model are shown in Figure 12.5. In order to get these results, a number of measures had to be eliminated from consideration, as was the case for the procedural memory model discussed above. Indeed, for both the payoff variability measures and the underweighting measures, the same set of conditions were removed (#2, #8, #21, and #24). For the loss rate effect conditions, only #31 and #32 were kept (rather than #36 and #40).

The overall conclusion is that neither the procedural memory nor the declarative memory model approaches the capabilities of the RELACS model.

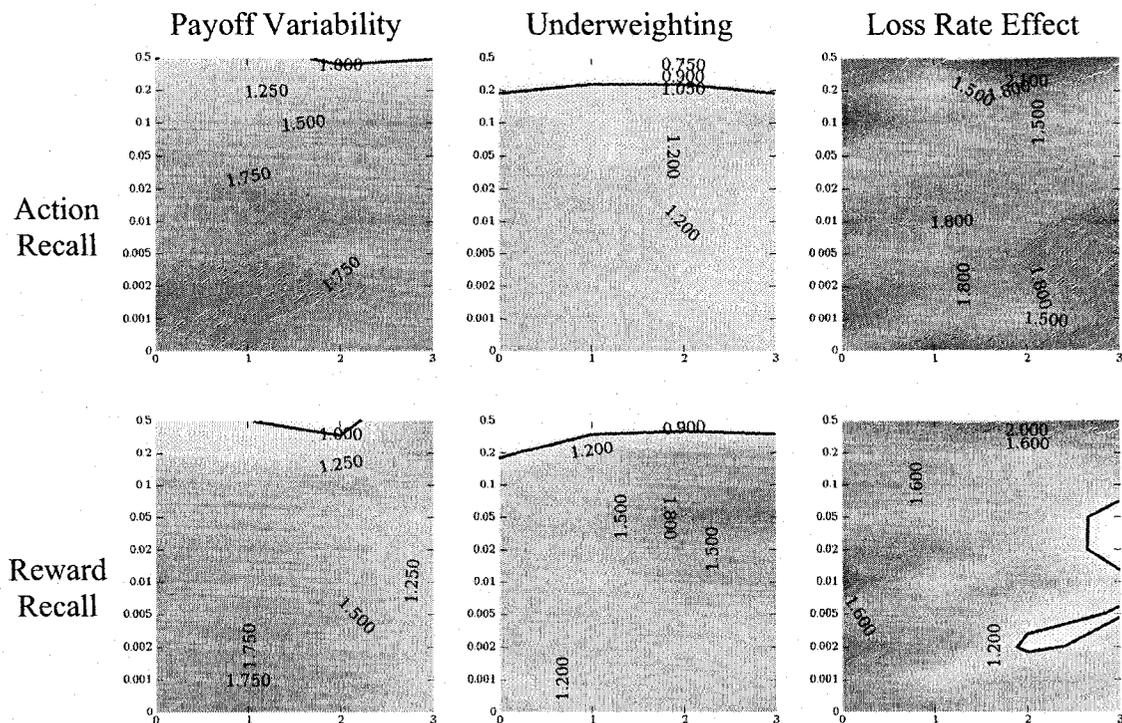


Figure 12.5: Relativized equivalence of the declarative memory ACT-R model for the three categories of repeated binary choice measures. X-axis indicates the lag, and the Y-axis shows the level of noise.

This comparison shows that RELACS's performance is significantly better than models that are directly available in the ACT-R cognitive architecture. While it is certainly possible that a more detailed model could be created in ACT-R that may be more accurate, at present there are no clear ways of doing so. By identifying this fact, the success of RELACS can be seen in its proper context. It is able to more completely account for this aspect of human performance, and less specific models have not been found which have similar abilities.

12.3 ACT-R and Peer Groups

ACT-R can also be used to create a model of popularity interactions within peer groups, the situation described in section 9.2. This creates an alternate model, completely

different from the one developed in section 10.2. As has been pointed out in various places throughout this thesis, it is as important to investigate different types of models as it is to investigate the effects of adjusting parameter settings. A Python ACT-R model provides a rather different sort of model to this same situation.

In creating this model, the ACT-R architecture imposes a major difference from the previous model. ACT-R makes the strong claim that exact *magnitude* is not something that is directly accessible to cognition. That is, a person might choose to be nice or not-nice, but cannot choose to be 1.7 nice or 2.48 not-nice. While it is possible to approximate magnitudes in ACT-R by using the architecture to construct a discrete magnitude scale, this would not represent the simplest and most direct use of the architecture. Therefore, the actions for the ACT-R model were restricted to being either nice or not-nice, with no magnitude component.

Another issue for the ACT-R implementation was how to keep track of how other people have behaved in the past. The ACT-R, procedural memory mechanism for learning the utility of actions could be used to do this in a way that is similar to the previous model. However, this would involve having separate and distinct procedural rules for every different person, which seems implausible. Instead, since people store information about individuals in declarative memory, the ACT-R declarative memory system was the basis for the model. This results in a qualitatively different type of model from our simple model.

The ACT-R declarative memory system does not have the ability to learn utility values

over time. However, it can be used to detect sequential dependencies in data generated across time, as was seen in the previous section in the declarative-memory based version of the repeated binary choice model. Following from this, the friendship interactions can be re-formulated as a matching game, where the goal is to predict if someone will behave positively or negatively to you, and then match that prediction with your own actions (i.e., if an agent expects someone to behave positively then they act positively toward them).

The ACT-R friendship model makes these predictions by using sequential dependencies to predict, from their previous n interactions, what an individual will do on this interaction. After each interaction, a record of what occurred, along with a record of the previous n interactions, is stored in the ACT-R declarative memory system as a *chunk*. Each time the same sequence of events is observed it strengthens the activation of that chunk in memory. Thus, chunk activation level reflects the past likelihood of a sequence occurring. When the system attempts to predict what will happen next, it can make use of its current knowledge of the previous n interactions with this individual to retrieve the matching chunk with the highest activation.

For example, if the opponent's last move was P (where P is positive and N is negative) and the model was set to use information from only the previous move (i.e., $n=1$; also termed *lag 1*), then there would be two possible matching chunks: PP (where the other person acted positively twice in a row) and PN (where the other person acted positively and then negatively). The activation levels of these chunks depend on the timing of previous situations where these sequential events occurred, according to the standard

ACT-R base level learning formula, described in chapter 11.

Thus, if PN had the highest activation (i.e. if this particular person had often responded negatively after being positive) then the model would respond negatively to match that expectation. The model would then observe what the other agent actually did and store a record of it. That is, if they actually responded positively, then the chunk indicating the sequence PP would be strengthened.

Importantly, all of these modelling details are generalized from earlier work, and were not designed specifically for this model. That is, the ACT-R model makes use of the established results on sequential dependency detection in the same way that it makes use of the established results on ACT-R modelling in general. The use of the ACT-R framework allows for the creation of a model that is simple to describe within the language of ACT-R, and at the same time inherits all of the properties of matching known psychological and neurological results.

To run the simulations, the same process was followed as with the previous model. Thirty ACT-R models were created (one for each individual), and they were allowed to interact over time. The CDC questionnaire was then applied, where each agent nominated three others that they liked and three that they disliked. This information was used to calculate the popularity classifications for each individual. Stability was determined by allowing further interaction and then repeating the questionnaire.

Since the only aspect of ACT-R used by this model is the declarative memory system, there is only a single parameter which controls this process. This is the noise inherent in

recalling previous events. This was kept at the canonical ACT-R value of 0.3, which has been established as suitable over a variety of models. This value is also the one that was found to work best for the game playing model (Lebiere & West, 1999), on which this model is based. Also, the model code was not customized in any way to fit the data. The friendship model is directly based on the game model. The ACT-R model can therefore be seen as an absolute prediction, with no possible room for parameter tweaking.

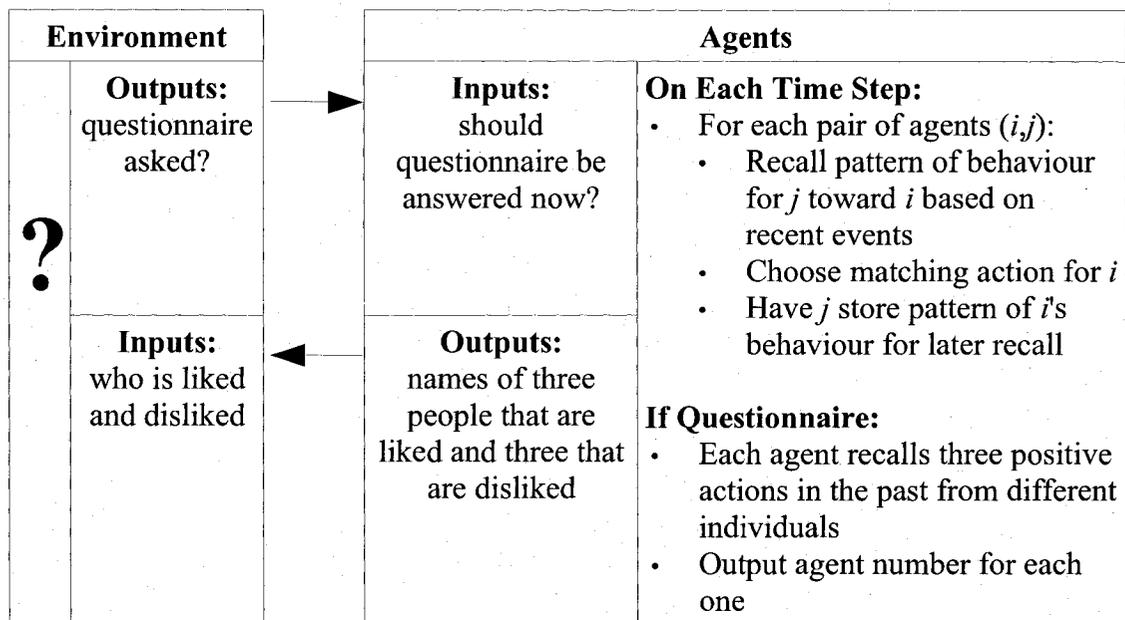


Figure 12.6: The ACT-R model for peer interactions

As before, the key dataset for comparison is the stability of category memberships. This was measured in an identical manner as with the computational model. Figure 12.7 shows the results compared to the expected real-world values. In this case, the only parameter varied was the amount of time between measurements, and all other parameters were kept at their canonical values.

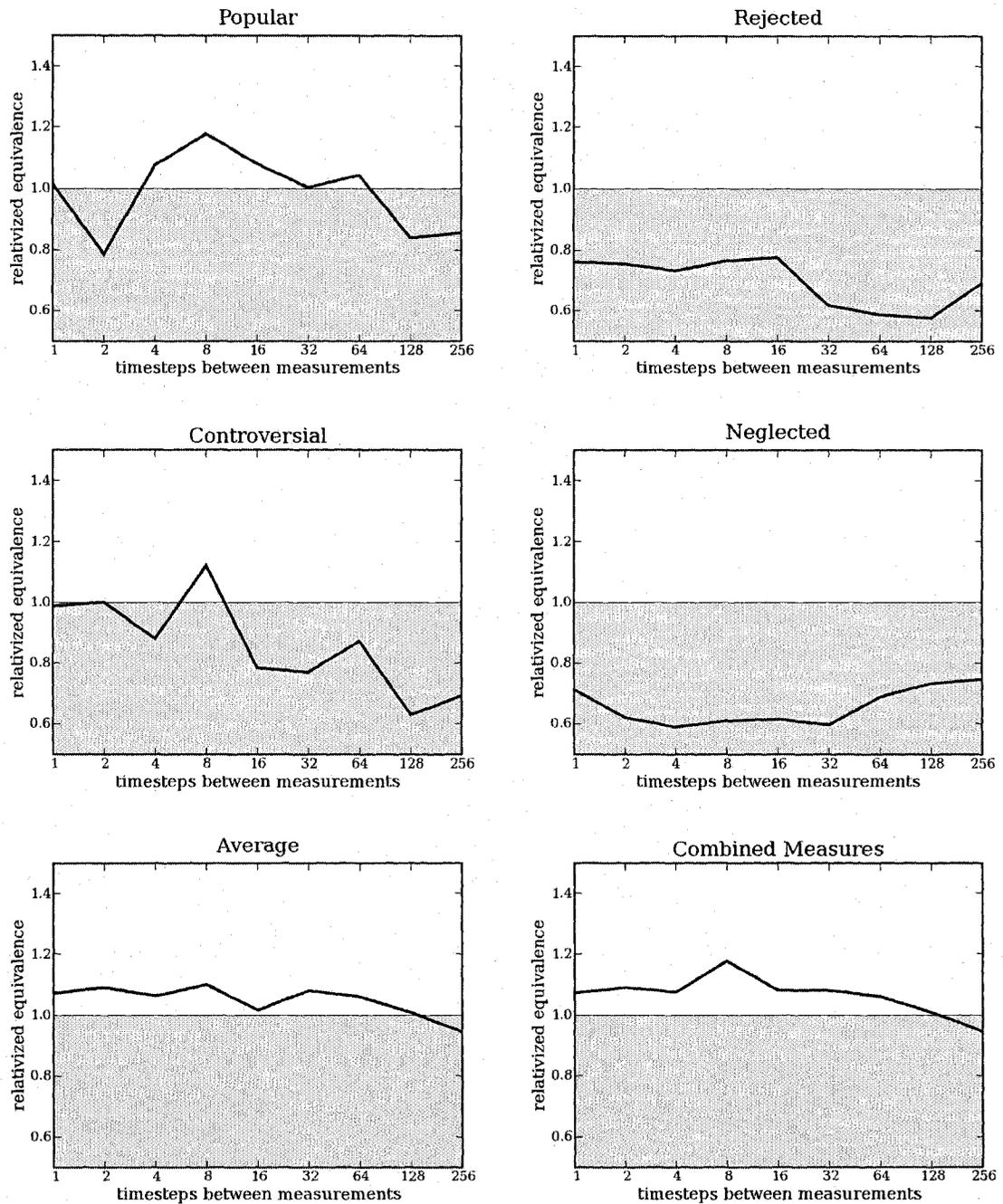


Figure 12.7: Individual and combined results for the ACT-R model of peer groups. Parameter settings inside the grey area are models which are equivalent to the human data on the CDC measures.

From these graphs it can be seen that the ACT-R model matches extremely well with the human results; in fact, it matches as well as the best of the earlier models. It is worth

repeating that *no model parameters were modified* to achieve this. It should be noted that the one measurement it has the most difficulty with is the stability of the Average category, which was the most problematic one for the mathematical model as well. This is remarkable, given that these are two completely different models of the same process.

12.4 Summary

12.4.1 Methodology

This chapter demonstrated the use of an existing cognitive architecture within the modelling methodology. This architecture, ACT-R, is the most well known and widely used architecture within cognitive science. This system allows for the quick creation of models that conform to a large body of pre-existing evidence. This makes it a natural place to develop new models of almost any cognitive phenomenon. The Python ACT-R system described in chapter 11 allows for the quick creation of such models and for them to interact with the environmental situations defined with CCMSuite. This makes direct comparisons between models possible.

The first example involved creating alternate models for the Repeated Binary Choice task. There are two natural ways this can be done in ACT-R. Either the choices can be handled by the procedural memory utility learning system (usually meant to indicate implicit knowledge), or it can be handled by the declarative memory system by remembering sequences of events in the past. However, none of these models were capable of achieving the level of equivalence of the RELACS model. This comparison indicates that the various cognitive strategies within RELACS capture the real

phenomenon more completely than is straightforward to do using ACT-R.

The second example showed an ACT-R model for interactions within a peer group. Here, the declarative memory and sequential dependency approach was again used. In this case, however, the resulting model was as good as the mathematical model presented in section 10.2. This is true even though the models differ in fundamental ways. For example, the ACT-R model only considers whether interactions were positive or negative and ignores how strong they were. However, both models successfully captured the empirically known pattern of category distributions and the stability of those categories. This means that the particular details of these models may not matter; instead, there may be underlying similarities in the models. Alternatively, if new measures are added, differences between the models may be identified.

12.4.2 Contributions

- RELACS was shown to provide a better account of human repeated binary choice behaviour (within the conditions identified) than the models based on ACT-R investigated in this chapter.
- An ACT-R model of peer group interaction, with all parameters set to the default canonical values, was created and shown to account for sociometric category and stability measures equally as well as the mathematical model presented in section 10.2.

13 Qualitative Models

There is an entire aspect of modelling that has been ignored throughout this thesis. All of the work discussed so far has dealt with situations where one or more models are being compared to a particular real system by means of particular quantitative values.

However, there are situations where computational models are used in cognitive science *but there is no particular real-world system for comparison.*

Many of the most well-known models in cognitive science fall into this category. The SHRDLU model of reasoning in a blocks-world (discussed in section 5.3) was not meant for comparison to a particular human performance. The famous Tit-for-Tat algorithm for the Prisoner's Dilemma, and the associated idea of the evolution of cooperation (Axelrod, 1981) also is not meant to exactly match human behaviour. Instead, the comparisons made between the model performance and the real world are much more qualitative. Here, the model is meant to exemplify and demonstrate some effect, rather than produce a behaviour which is numerically equivalent to that of the real system.

Technically, it may be argued that in such a case, the computer programs should not be considered models. After all, in order for a program to be a model, there must be something that it is being modelled. However, this type of endeavour is common within cognitive science, and so it is worth determining how the methodology developed in this thesis can be applied to such situations.

The approach taken here is to treat these as *qualitative models*. That is, the intent is still to have some form of comparison between the model and reality, but this is not a

numerical comparison. Instead, the comparison is more about what patterns of behaviour are evident in the model, and whether those patterns are similar to those seen in the real world. These are much weaker comparisons than those considered in the rest of this thesis, but they are certainly a useful initial approach to understanding a cognitive phenomenon.

To demonstrate this, two examples of qualitative modelling are now considered. The first starts with the models of peer group interaction discussed in sections 9.2, 10.2, and 12.3 and uses these models to investigate the effects of individual differences among the interacting agents. This produces qualitative predictions about what sorts of internal differences can lead to individuals being more likely to be classified as Popular or Controversial, for example. This demonstrates one way a model can be used in an exploratory manner once it has been previously established using this methodology.

The second example shows the creation of an entirely new model as an *existence proof* for a particular idea about the formation of representations. Here, a particular theory about how cognitive agents make use of structures in the world inspired a simple learning model. When run, this system is able to learn to modify its environment in such a way as to make it easier for it to perform a foraging task. Since this learning model is so robust (i.e. it works well across a wide range of parameter settings), this raised the possibility of a similar learning system being used to modify the *internal* environment of the agent. In this new situation, the agent was able to learn to remember certain aspects of its environment, forming “proto-representations”. By exploring the parameters of this model, the limits and constraints of this learning process are identified.

13.1 Model Exploration

In this section, the mathematical and ACT-R peer group interaction models described in sections 10.2 and 12.3 are modified to introduce differences among the individuals within the model. For example, some individuals may interact more often with others, or they may be biased to behave positively or negatively, or they may be faster or slower at learning about others. Given these variations, individuals may be more or less likely to end up classified as Popular, Rejected, Neglected, Controversial, or Average.

However, there is no direct measure which allows for a numerical comparison between the behaviour of this model and the behaviour of real children. This is because the model adjusts internal aspects of the individual which are not directly measurable. If it turns out that individuals classified as Controversial have a learning rate that is 1.36 standard deviations above the average, it is very difficult to compare this to real human data.

However, instead of examining the precise quantitative value, the approach taken here is to identify the patterns and trends in the data and compare these trends to known real-world trends. In this case, the key point is just that Controversial individuals tend to be faster at learning than average.

For each type of individual variation, the same analysis process was used. Changes were made to the model so that a numerical value for each individual encoding the change being investigated. This value was set randomly for each individual, using a Gaussian distribution. The standard deviation of this distribution was increased to be as large as possible while still maintaining the equivalence of the model and the distribution and stability data used previously. This ensured that the changes would have some effect, but

not so much to ruin the model's equivalence. The *effect size* is then measured. This is the number of standard deviations above or below the mean values of the individual difference value are for the individuals within each category, as compared to the Average individuals. This is a technique used by Newcomb, Bukowski, and Pattee (1993) to relate individual variations in personality metrics to CDC classifications.

It should be noted that these investigations were done after the development of the model, and the model was not developed with these variations in mind. This is an example of taking a model that was developed in one domain (the prediction of CDC stability data) and using it in a new domain (the effect of individual differences on CDC classification). This sort of model extension is the primary method that science uses for determining if a model or theory captures an important and useful aspect of the phenomenon in question. Any model that cannot be extended past the situation it was designed for may be merely descriptive, and thus not useful for understanding the fundamentals of the situation.

The mathematical model and the ACT-R model are analyzed separately here.

Introducing these individual variations into the model must be done differently in each case, but similar sorts of variations are introduced. This research has been previously published (Stewart, West, & Coplan, 2007).

13.1.1 Individual Variations in the Mathematical Model

As a reminder, the mathematical model of peer group relations was described in section

10.2. The basic algorithm is as follows:

1. Let $a[i,j]$ be the amount person i likes person j

2. For all pairs of individuals i, j :
 - a. Determine how i behaves toward j as $b[i, j] = G(a[i, j], 1)$
 - b. Determine how j behaves toward i as $b[j, i] = G(a[j, i], 1)$
 - c. Update how much i likes j : $a[i, j] \leftarrow a[i, j] + r(b[j, i] - a[i, j])$
 - d. Update how much j likes i : $a[j, i] \leftarrow a[j, i] + r(b[i, j] - a[j, i])$
3. Repeat step 2 for as much time as is desired

After time has passed, the CDC classification system can be used to categorize each individual as Popular, Rejected, Neglected, Controversial, or Average. This is done by having each individual nominate three others that they like and three that they dislike. The rules for categorization were given previously in Table 9.6.

13.1.1.1 Interpretation Bias

The first individual variation deals with the Hostile Attribution Bias. This refers to the fact that certain individuals tend to interpret the actions of others in a more negative light than is intended. Research has shown that Rejected children (particularly those who are aggressive) are more likely to assume malevolent intent when they are faced with ambiguous social cues (Crick & Dodge, 1994; Dodge, Lansford, Burks, Bates, Pettit, Fontaine, & Price, 2003). To model this, an individual difference was inserted in the model by adjusting the formula used in steps 2c) and 2d) above.

$$a[i, j] \leftarrow a[i, j] + r(b[j, i] + B[i] - a[i, j])$$

In this new formula, $B[i]$ is an individual interpretation bias. This can be either positive or negative. Agents can thus be biased to be either overly negative ($B[i] < 0$) or overly positive ($B[i] > 0$) in how they interpreted the actions of others.

This simulation was then rerun over 1000 groups of 30 agents each. Each agent had a value of $B[i]$ chosen from a Gaussian distribution with a deviation as large as possible while still matching the aforementioned results. After 500 simulation iterations, CDC classification was performed and the effect size was measured.

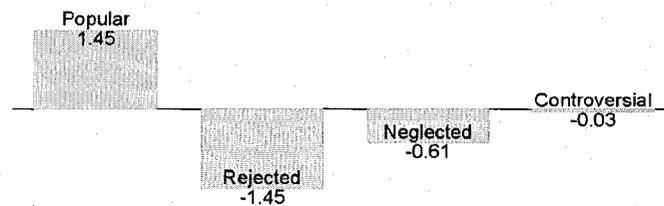


Figure 13.1: Effect size for mathematical model when adjusting interpretation bias.

The results in Figure 13.1 show that the simulation predicts that Rejected individuals tend to be overly negative in interpreting the behaviours of others, while the Popular ones tend to be overly positive. This is exactly in accord with the Hostile Attribution Bias.

Interestingly, the model also generates a novel (and testable) prediction, as Neglected individuals also tended to be somewhat negative in their interpretations. Importantly, the particular numerical value of this prediction is not the point of this qualitative model.

Instead, it is the pattern of results which would be expected to appear in the real world.

13.1.1.2 Behaviour Bias

Another well-known finding is that Popular children tend to have good social skills whereas Rejected children tend to have poor social skills. (e.g., Coie, Dodge, & Kupersmidt, 1990; Newcomb, Bukowski, & Pattee, 1993; Parkhurst & Hopmeyer, 1998). That is, popular children tend to express themselves in ways that mitigate bad feelings whereas rejected children tend to express themselves in ways that exacerbate bad

feelings. To reflect this in the model, the mean of the Gaussian used in steps 2a) and 2c) was biased. Specifically, a positive value created a bias toward behaving nicely and a negative value created a bias toward behaving badly. Figure 13.2 displays the results and shows that the manipulation had the intended effect. Interestingly, the Neglected agents were again shown to be somewhat similar to the Rejected agents. Note also that varying the ability to express oneself socially produces approximately the same distribution of effects that varying the ability to accurately assess another persons' intentions (i.e., the hostile attribution bias). This makes sense as the end effect is essentially the same, that is to bias the valence of the behavior.

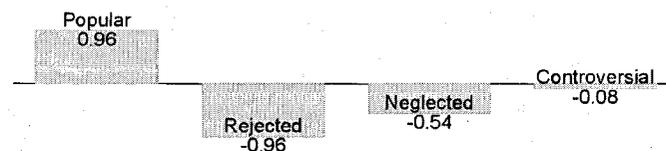


Figure 13.2: Effect size for mathematical model when adjusting behaviour bias.

13.1.1.3 Interaction Rate

The third effect investigated was that Neglected children have been shown to interact with their peers less frequently than average children (Dodge, Coie, & Brakke, 1982; Coie & Dodge, 1988). To reflect this in the model, an interaction probability for each agent was added. The percentage chance for two agents interacting was determined by multiplying their interaction probabilities together. As illustrated in Figure 13.3, this manipulation was successful in capturing the effect. It also produced the unexpected effect that a high level of interaction was associated with being Controversial.

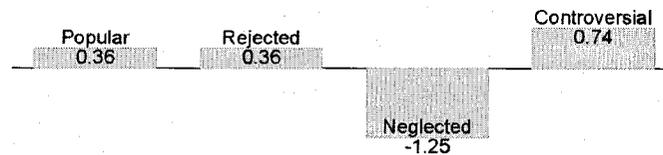


Figure 13.3: Effect size for mathematical model when adjusting interaction rate.

13.1.1.4 Behavioural Consistency

Next, the finding that controversial children tend to display a combination of positive and negative social behaviours (e.g., Coie & Dodge, 1988) was examined. This lack of consistency was modelled by varying the standard deviation of the Gaussian distribution in steps 2a) and 2b). The results, displayed in Figure 13.4 support the interpretation that Controversials tend to be highly variable in their behavior. The results also revealed an unexpected effect in which Neglected individuals tended to be more reliable in their behavior (lower variability). Note also that making children more variable produces approximately the same distribution pattern as making them interact less frequently. This makes a certain amount of sense, since highly variable behaviours would result in more actions that cancel each other out, which would approximate the effect of simply not interacting as much.

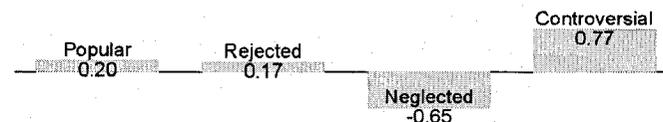


Figure 13.4: Effect size for mathematical model when adjusting behavioural consistency.

13.1.1.5 Predisposition

The next variation examined was the initial value for $a[i,j]$. For this simulation some agents started off more predisposed to liking everyone (a high value of $a[i,j]$ for all j), and

others more predisposed to disliking everyone (a low value of $a[i,j]$ for all j). This was meant to represent the effect of learning, before entering the group (e.g., family experiences or other peer group experiences). Thus there were no individual differences during the simulation; only the stored information at the beginning was different. The idea behind this was to see if the model displayed sensitive dependence on its initial conditions, a common feature in dynamic systems. The results of this simulation are displayed in Figure 13.5. Note that the results were very similar to the results of the hostile attribution simulation and the social skills simulation. All of these showed an expected association between strong positive biases and being popular, and strong negative biases and being rejected. They also all showed the unexpected association between moderate negative biases and being neglected.

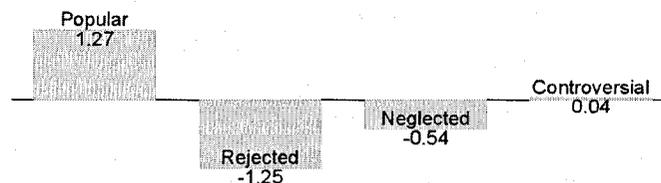


Figure 13.5: Effect size for mathematical model when adjusting initial predispositions.

13.1.1.6 Learning Rate

The final individual variation analyzed involved varying the parameter r . This investigates how having different adaptation rates would affect one's eventual CDC categorization. The purpose of this was to explore the effects of individual differences on the only cognitive parameter in the model. The results, displayed in Figure 13.6, were interesting, in that they were very similar to the results of the interaction probability simulation and the behavior consistency simulation. The most straightforward interpretation of this is that having a high learning rate or having many interactions can

have the effect of making one appear to be more variable in behavior.

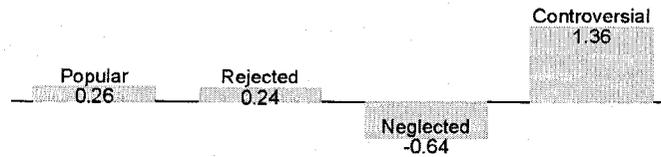


Figure 13.6: Effect size for mathematical model when adjusting learning rate.

13.1.1.7 Patterns of Individual Variation

The results on the effects of individual differences can be divided into two categories, which are shown in Figure 13.7. The key point to observe here is that there are only two major effects caused by varying these six different individual traits. In the left-hand graph, it can be seen that Popular individuals tend to be those who interpret others actions in a positive light, tend to behave better toward others, and who start with a positive attitude toward others. Rejected individuals tend to be negative in all of these traits. This is not a particularly surprising result. What is surprising is that these same three traits are also grouped together for Neglected individuals and for Controversials. People who are Neglected by their peers tend to be slightly negative in these three traits, and Controversials are the same as Average people in these traits.

The other three traits are also closely grouped, as shown in the right-hand graph.

Interacting more often, having more random deviations in one's actions, and changing one's beliefs quickly in response to others' actions are all traits common in Controversial people. Neglected people, on the other hand, tend to be low in those same three traits.

Furthermore, both Rejected and Popular people tend to be moderately above the average.

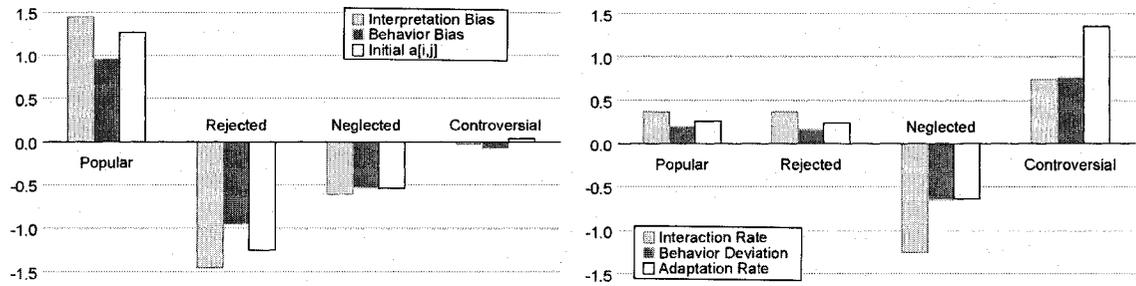


Figure 13.7: Summary of individual difference effects for the mathematical model.

It is important to stress that the precise numerical values of these results are not an intended prediction of this model. However, the over pattern is highly regular. Since this model has been established as equivalent to the human results in a variety of ways, one can expect a certain amount of similarity between these variations of the model and variations in the real interacting people. Certainly, many of the effects identified here are highly consistent with known psychological effects (such as the Hostile Attribution Bias). None of the results seem to contradict known findings, and can be interpreted as predictions of the model. This allows the model to be used to create new real-world research which can lead to the confirmation or disconfirmation of these predictions.

13.1.2 Individual Variations in the ACT-R Model

The ACT-R model is more constrained than the mathematical model because experimental research with ACT-R has uncovered specific parameter settings that tend to lead to models that predict human behavior well (the canonical parameter values). However, it is still possible to introduce cognitively based individual differences by adjusting the architecture parameters. One way of doing this is to adjust the ACT-R declarative memory instantaneous noise parameter, which adjusts the variation in the

logistic noise in the activation equation discussed in chapter 11. This would not affect learning but it would make the retrieval process noisy, such that lower noise would make it more likely that the chunk with the highest activation value is chosen and higher noise would make it more likely that a different chunk is chosen. Essentially this is a way to make the model's behaviour more variable.

Alternatively, another way of adjusting the ACT-R model is to bias it. This can be achieved by asymmetrically adjusting the number of times the chunk representing what just happened is put into declarative memory. A positive bias is created by storing positive events in declarative memory twice and negative events once. Conversely, a negative bias involves storing negative events in declarative memory twice and positive events once. Conceptually, this manipulation can be related to people paying more attention to certain outcomes. It should be noted that ACT-R makes the strong claim that models must perform a whole number of repetitions, so there is no mechanism for increasing the number of presentations of activation of a chunk by a fractional amount.

In addition to these two manipulations it is also possible to directly recreate two manipulations used in the mathematical model. The initial memory bias (predisposition) can be recreated by raising the activation values of chunks associated with predicting unfriendly behavior or friendly behavior. This is accomplished by adding a fixed value to the activation equation for each relevant chunk. The interaction rate between individuals can be implemented in the same way as was used for the previous model, by having each individual have a set probability of interacting.

These four changes to the ACT-R model correspond to the behavioural consistency, behaviour bias, predisposition, and learning rate adjustments seen in the previous section. The results of these variations can be seen in Figure 13.8. What is most striking about these results is that the pattern is exactly the same as that seen in the mathematical model. The fact that this pattern appears in two highly different models indicates that there is an underlying similarity that both are capturing. This makes it even more reasonable to suspect that a similar pattern may be found in the real world.

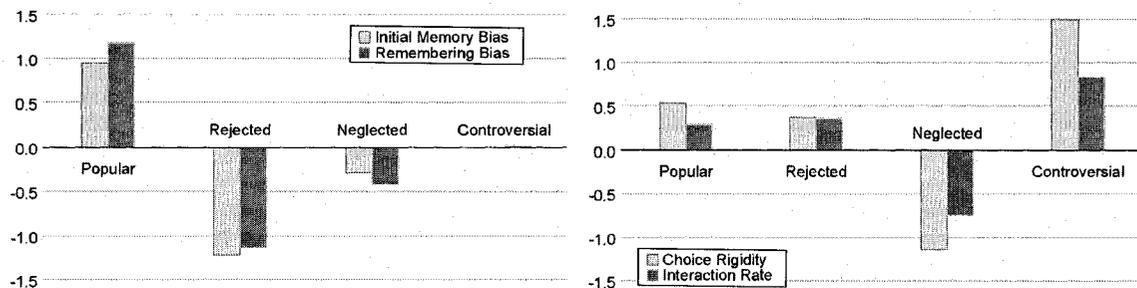


Figure 13.8: Summary of individual difference effects for the ACT-R model.

13.2 Existence Proofs

In the previous section, a model that had been established to be equivalent in one domain (the stability of popularity categorization) was used to explore another domain (the effects of individual differences). In this section, the goal is to investigate whether a particular cognitive model is *capable* of performing some particular task. This is an example of using a computational cognitive model as an *existence proof*: a demonstration that it is possible for a particular cognitive phenomenon to be exhibited by a particular type of model. Given the complexities of cognitive models, it is quite often unclear what behaviours will be exhibited until such a demonstration is made.

This sort of exploratory work is often associated with the creation of new overall theories of how a particular cognitive behaviour might arise. Creating such an existence proof model is meant to indicate that the theory could, in fact, account for a phenomenon. However, at this initial early stage, there is generally no attempt to match precise numerical values. Instead, it is merely the overall qualitative behaviour of the model that is of interest.

To demonstrate this process, a qualitative model of epistemic structure formation is presented here. This work stems from a collaboration with Dr. Sanjay Chandrasekharan, who developed the underlying theoretical framework, and portions of this section are based on a previous publication (Chandrasekharan & Stewart, 2007).

13.2.1 Learning to Create Epistemic Structures

The core idea being demonstrated by this model involves *epistemic structures*. These are physical structures in the world that are created in order to simplify particular tasks. This is a widely observed natural phenomenon. Wood mice distribute small objects, such as leaves or twigs, as points of reference while foraging. This behaviour is also seen in laboratory conditions using plastic discs (Stopka & MacDonald, 2003). Red foxes use urine to mark food caches they have emptied. This marking acts as a memory aid and helps them avoid unnecessary search (Henry, 1977, reported in Stopka & MacDonald, 2003). The male bower bird builds colourful bowers (nest-like structures), which are used by females to make mating decisions (Zahavi & Zahavi, 1997). Ants drop pheromones to trace a path to a food source. Many mammals mark their territories

(Bradbury & Vehrencamp, 1998). Bacterial colonies use a strategy called quorum sensing to know that they have reached critical mass (to attack, to emit light, etc.). This strategy involves individual bacteria secreting molecules known as auto-inducers into the environment. The auto-inducers accumulate in the environment, and when they reach a threshold, the colony moves into action (Silberman, 2003). Finally, this is also a common behaviour exhibited by human beings, as evidenced by written notes, street signs, uniforms, and so on.

Given this ubiquitous cognitive phenomenon, one might wonder how it arises. In humans, the naïve assumption that the creation of these structures is driven by explicit, high-level reasoning seems reasonable. However, it is not clear how this is performed in other creatures, and it may be that whatever mechanism is responsible for this in other creatures is also active in humans. The question, then, is what possible low-level mechanisms could lead to such behaviour.

One possibility is that the construction of epistemic structures is learned via evolution. In this case, creatures would simply be hard-wired to build certain structures at certain times, and make use of them in different ways. If such a system provided evolutionary advantage (by making it easier to find food, find mates, or avoid predators, for example) then it could become an established behavioural trait. However, this approach does not allow for a particularly flexible solution, as it would mean that learning to create these structures would have to happen over an evolutionary time scale.

Instead, the core idea developed by Chandrasekharan (2005; Chandrasekharan & Stewart,

2007) is that these structures can be learned during the lifetime of the creatures by tracking the *tiredness* or amount of effort needed to perform a task. For this to work, the organisms must sometimes generate random structures in their environment, and be able to keep track of the amount of effort expended. If there is then a mechanism which can identify situations where particular structures were created and effort was reduced, then this can be used to increase the likelihood of making such structures.

This description of the situation reduces it into a classic reinforcement learning paradigm. Many actions are available to the organism, some of which involve creating structures. The amount of effort expended is the reward mechanism. Any reinforcement learning strategy could then be used to learn that creating structures in some situations is beneficial (i.e. when those structures can be detected later and used to reduce effort).

13.2.1.1 Environment 1: Route Finding

While the theoretical idea of learning to create epistemic structures based on tiredness feedback seems somewhat plausible, the idea is more convincing if an actual example can be demonstrated. This involves creating a computational model of some situation, creating an agent that can learn in such a way, and determining if, in fact, the agent is able to make use of these structures to improve at whatever task it should be performing.

Importantly, there is no particular real-world situation being modelled here. Instead, the intent is that the implemented situation is merely of the same kind as the real-world situations, and that the learning algorithm used by the agent follows the theoretical framework described above. But the exact task, and the exact learning mechanism are

not meant to be modelling particular real-world tasks or real-world learning system.

The chosen task is based on foraging behavior: navigating from a home location to a target location and back again. The environment consists of a 30x30 toroidal grid-world, with one 3x3 square patch representing the agent's *home*, and another representing the *target*. This target can be thought of as a food source, to fit with the analogy to foraging behavior. Since most real-world epistemic structure generation involves group interaction, multiple agents will be attempting to forage simultaneously. The epistemic structures that can be generated are actions that make particular parts of the environment more *home-like* or for *target-like*. It should be noted that this is a more complex environment than has been considered previously in this thesis. This is due to the necessity of having a complex world for the agent to interact with.

One way to think about this process is to consider the pheromone-following ability of ants. Common models of ant foraging (e.g. Bonabeau et al, 1999) postulate the automatic release of two pheromones: a home pheromone and a food pheromone. The ants go toward the home pheromone when they are searching for their home, and they go toward the food pheromone when foraging for food. This matches these two actions in our agents. The home pheromone would be an example of a home-like structure in the ant environment.

At any given time, an agent can do one of five possible actions. The first and most basic of these is *move randomly*. This consists of either going straight forward, or turning to the left or right by 45 degrees and then going forward. The agent does not pick which of

these three possibilities occurs (there is a 1/3 chance of each).

The next two actions involve the basic sensory abilities of the agent. These are exactly like the first action, but instead of moving randomly, the agent could move toward whichever square was sensed to be the most *home-like* (or the most *target-like*). Initially, the only things in the environment that are home-like or target-like are the home and the target themselves.

The fourth and fifth possible actions provide for the ability to generate these home-like and target-like structures. In the standard ant models, this could be thought of as the releasing of pheromones. However, in this simulation, the ability to modify the environment is something the agents can do *instead* of moving around. That is, this generation process requires time and effort. One way to interpret this is to think of an action that a creature might do which inadvertently modifies its environment in some way. Examples include standing in one spot and perspiring, or urinating, or rubbing up against a tree. These are all actions which modify the environment in ways that might have some future effect, but do not provide any sort of immediate reward for the agent.

In order to choose an action to perform in a meaningful way, each agent must have some sensory capability that can serve to direct the choice of actions. These agents had four sensors, two external and two internal, to identify their current situation. The two external sensors sense how home-like and how target-like the current location is (digitized to 4 different levels). One internal sensor indicates whether the agent has been to the target yet (yes or no), and the other indicates how long it has been since the agent

generated a structure in its environment (up to a maximum of 5 time units). This is all that the agents can use to determine which action to perform. This configuration gives each agent 192 ($4 \times 4 \times 6 \times 2$) possible different sensory states.

In this simulation, a creature must expend extra effort to generate structures in the world. An agent that does this will be efficient only if the effort spent in generating these structures is more than compensated for by the effort saved in having them in the world. Moreover, these are not permanent structures. The agents' world is dynamic and the structures do not persist forever. The home-likeness or target-likeness of the grid squares decrease exponentially over time. Furthermore, these structures also spread out over time. A home-like square will make its neighbouring squares slightly more home-like. This can be considered similar to ant pheromones dispersing and evaporating, or leaf/twig piles being knocked over and blown around by wind or other passing creatures.

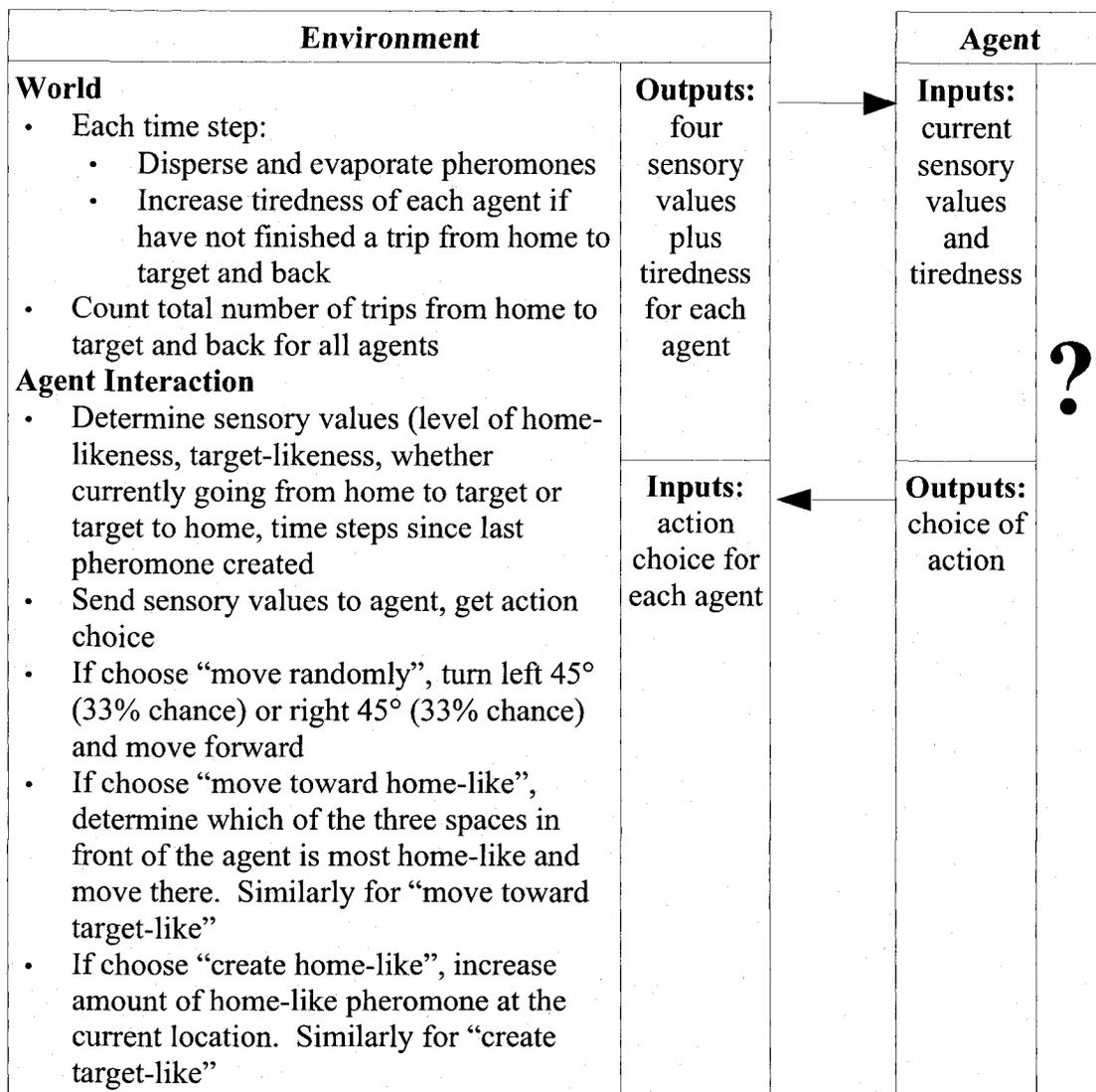


Figure 13.9: Specification for the simulation environment for exploring epistemic structure generation.

13.2.1.2 The Q-Learning Model

From the environment specification in Figure 13.9, it can be seen that each agent must, each time step, choose one of the five possible actions to perform. The only inputs available to make this choice based on are the four sensory values, plus the tiredness level. The goal of the agent should be to reduce tiredness as much as possible. Since

tiredness always increases, except when a trip is completed from the home to the target and back, the only way it can do this is by making these trips more quickly. However, the agent has no direct way of knowing what actions will lead to the target and home.

Initially, there will be no structures in the environment at all. This means that trying to go toward the most target-like structure will result in random motion. However, if the agents are able to learn to periodically create target-like structures while searching for the target, and to create home-like structures while searching for the home, then future trips can be made simply by following the trail of these structures.

There are a number of possible learning systems that might be capable of performing this task. Since the phenomenon is wide-spread and seen in many simple creatures, the decision was made to attempt extremely simple learning models, rather than something as complex as ACT-R. The most natural candidate for this is the standard Q-Learning reinforcement learning system (Watkins, 1989). This requires a sensory state as an input, and chooses a particular action based on a reward signal (in this case, the tiredness value).

It must be stressed here that this implementation does not presume any sort of long-term planning on the part of the agents. Given a set of available actions, they will choose these actions in a purely reactive manner (i.e., based entirely on their current sensory state).

They do not initially have any sort of association between the action of making home-like structures and the action of moving toward home-like things. Any such association must be learned over time based on experience.

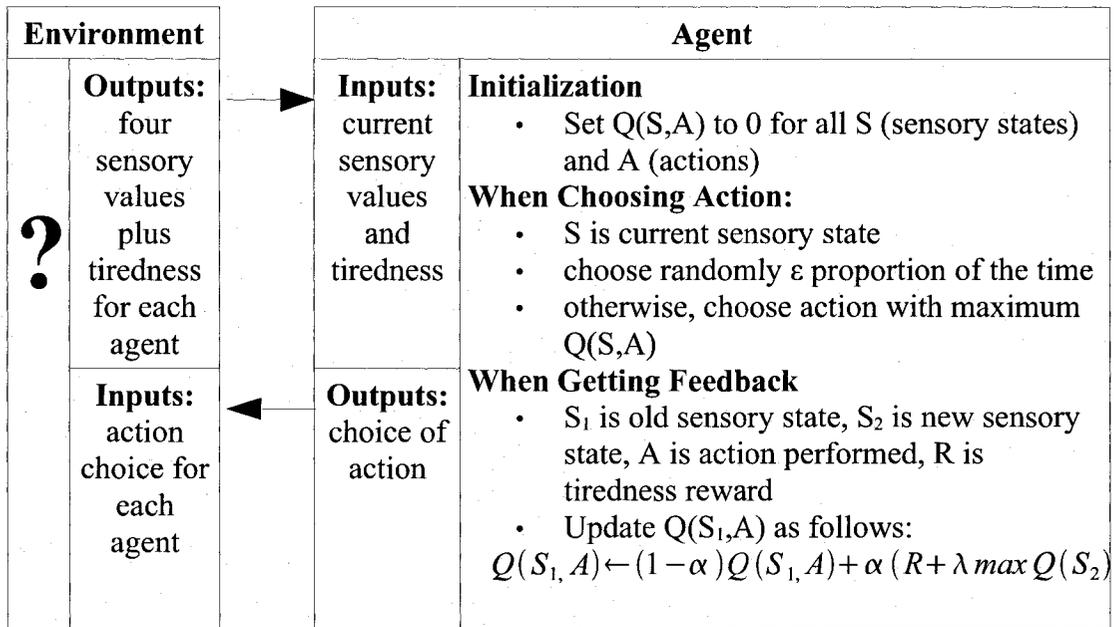


Figure 13.10: Overview of the Q-Learning model for learning to use epistemic structures for foraging.

13.2.1.3 Results

When the Q-Learning agents are allowed to run in the foraging world, they initially behave randomly. They spread out over the world quickly, occasionally creating structures. After a while, these structures are created in a more useful manner, as they are concentrated along the path from the target to the home. The agents are then able to use these structures to more quickly travel from one to the other.

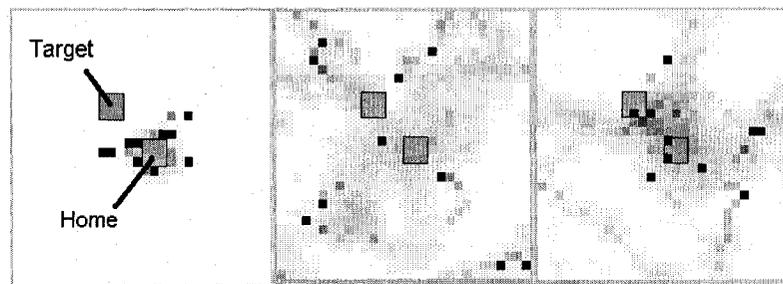


Figure 13.11: Agents foraging after 10, 100, and 300 time steps. Black dots are agents. Shading indicates structures that have been created.

The purpose of this model is to demonstrate that the agents are able to learn to make use

of epistemic structures in order to simplify the task of travelling to the target and back home. In order to demonstrate this, two things must be true. First, the agents should improve over time at their foraging task, making more trips in less time. Second, they should perform this task better than agents which do not have the capability of creating epistemic structures. This can be demonstrated by running a version of the model which removes the creation actions. These results are shown in Figure 13.12. Both of the expected effects are evident. The learning mechanism is able to learn to perform significantly better when it is capable of creating structures in the environment. Interestingly, there is an initial drawback to structure generation, as seen at the very beginning of the graph. It is not until more than 100 time steps have passed that the constructing agents show improvement. This is likely due to the amount of time wasted initially when learning what structures are useful to create in what situations.

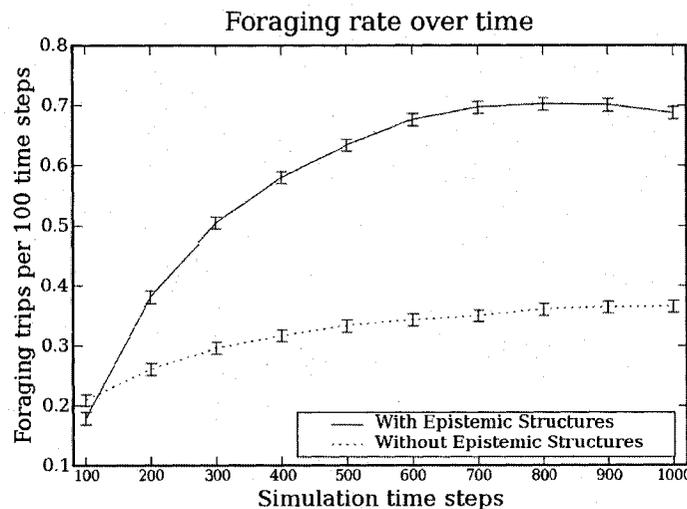


Figure 13.12: Foraging rate for the epistemic structure model.

There are numerous investigations that can be made to determine how robust this effect is. Q-Learning has two parameters (the learning rate α and the temporal discounting

factor λ). Also, the particular organization of the environment is important. What happens if the target is farther away from the home? This sort of variation gives a useful insight into what sorts of situations this approach to learning may be applicable to. The results of adjusting this distance are shown in Figure 13.13. This shows that there is a decreasing advantage to structure generation as the distance increases. This reflects the difficulty of the task, and indicates that this basic version of this learning procedure should not be expected to be effective for long-range tasks.

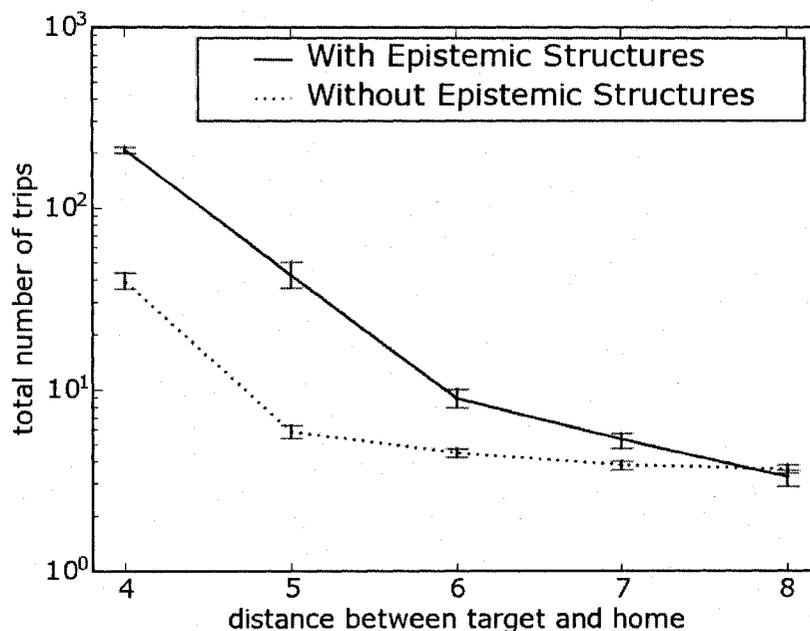


Figure 13.13: The effect of varying the distance between target and home.

The overall result is that this simple learning mechanism can successfully take advantage of an ability to create structures in the world. This mechanism is relatively simple, and does not require a complex cognitive architecture to support it. It is a general process that could be applied to any situation, given a sensory state, a set of possible actions, and a reward measure that indicates how successful the agent is at a task. This model is thus

an existence proof demonstrating that agents can learn this sort of task within their lifetimes, and do not need to rely on either evolutionary mechanisms or a complex set of high-level reasoning skills.

13.2.2 Learning to Create Internal Structures (Proto-Representations)

Given the success of the epistemic structure model for demonstrating that Chandrasekharan's theoretical framework is at least possible, a further existence proof was developed that pushed the theory even further. Here, the attempt was to see if this same mechanism that was responsible for creating and making use of *external* structures could also be used to create and make use of *internal* structures. That is, since these epistemic structures can be thought of as external *representations*, it may be possible that this same learning system, combined with the ability to create and detect internal states, could lead to the creation of internal structures in the same way.

These structures would clearly not be full-blown conceptual representations. They would not be, for example, compositional, any more than the ants' pheromones are compositional. However, they would be standing in for something in the external world, and could be considered precursors to full representations. For this reason, they have been termed *proto-representations* (Chandrasekharan & Stewart, 2007).

13.2.2.1 Environment 2: Remembering State

To determine whether it is possible for this mechanism to work, a variation of the foraging model from section 13.2.1.1 was created. In that original environment, one of

the sensory values available to the agent was whether it was heading for the target or for the home. This value is needed so that, while the agent is partway between the two, it is able to determine which way it should go. For example, it may learn that when it is trying to find the target, it is useful to choose the action of moving toward target-like structures (and occasionally create structures).

This memory of whether the agent is currently seeking the target or the home is exactly the sort of internal structure that might be learnable by the system. That is, instead of having a built-in mechanism which keeps track of this, the agent might be able to *learn to remember* which way it is going. It could choose to perform a “remembering” action, and the internal structure created by this action could be used later to determine which way the agent should be going.

To simplify this task as much as possible, it was decided to not attempt to have the agent learn to use both the external epistemic structures and the internal structures at the same time. Instead, the agent would be able to choose to move directly toward the target or the home without requiring the structural markings. However, it would need to learn to remember which way it was supposed to go.

Like the previous model, there are five actions available to the agent. Three involve motion: moving randomly, moving toward the home, and moving toward the target. The other two are for creating structure. This involves placing a value in an internal memory. In this model, the agent can choose to store either a zero or a one. This stored value will be available to its internal sensors, so it can be used by the Q-Learning algorithm to

choose actions.

The most direct way of implementing this does not work. That is, if the internal memory is implemented as a simple data store which values are placed in, then the agent is not able to perform better than an agent without the ability to remember. This failure led to a closer examination of how this internal memory could be implemented so as to be more like the external epistemic structures, which were able to be learned. Three aspects of that may be needed for successful representation were identified:

- 1) Information needs to be stored associated with particular context. When the epistemic structures are created, they are at a particular *physical location*. Similarly, internal structures could be associated with particular sensory states.
- 2) Information needs to be recalled in a particular context. Only the epistemic structures that are at the same location as the agent can be sensed at any given time. Similarly, internal structures may be organized so as to only be usable in contexts similar to how they were stored.
- 3) Information needs to spread and change gradually. The epistemic structures do disperse and slowly evaporate, but only in a gradual manner. Internal structures may then be organized so that data stored in one context would be available in other, similar contexts, and any new data stored should only make small changes to the memory.

A well-studied mechanism that has exactly these characteristics is the standard feed-

forward neural network trained by back-propagation of error (Rumelhart, Hinton, & Williams, 1986). Such a system was chosen for the internal memory mechanism. Just as the first simulation had agents with mechanisms for creating and sensing external structures, in this experiment we the agents can choose an action which stores data in this network, and a mechanism for sensing the current output of the network. This network thus plays the same role as the external environment in the first experiment, becoming an “internal environment” (Dennett, 1975; Hills, 2006). In this case, a the network has three input nodes, three hidden layer nodes, and one output node. The three input nodes indicate the current sensory state, and the output node indicates the value currently associated with this current sensory state.

It should be noted that there is a subtle recursion happening in this model. One of the components of the agent’s sensory state is the output of the neural network (the internal environment), but that output is itself dependent on the current sensory state, since that is an input to the neural network. This means that what the agent remembers *is dependent on what it is currently remembering*.

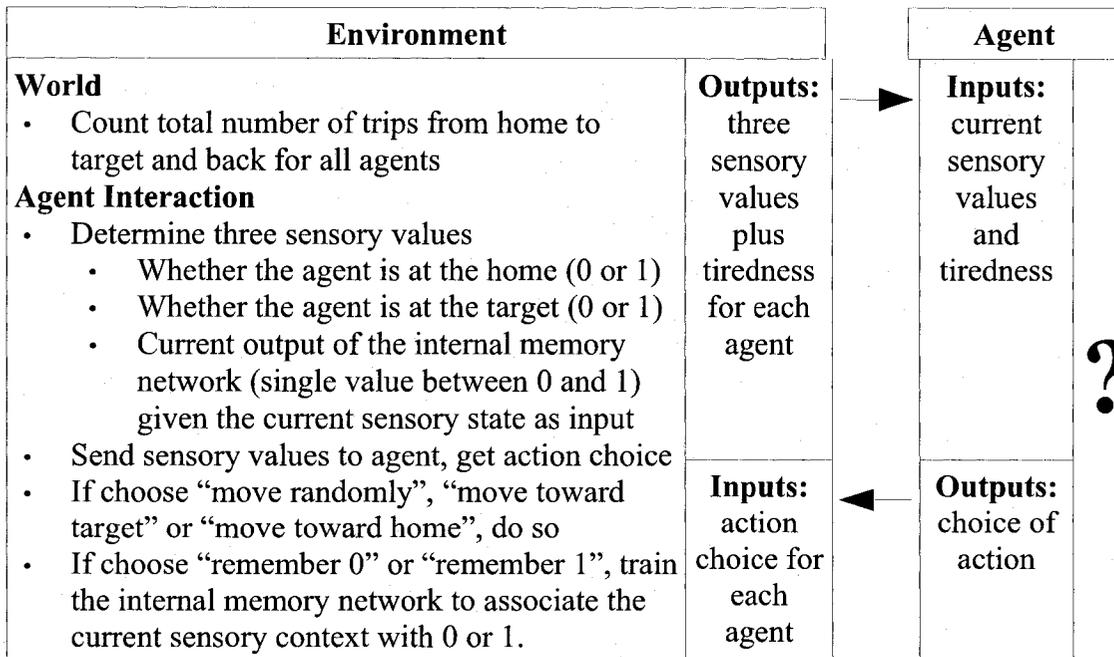


Figure 13.14: Specification for the simulation environment for exploring internal structure generation.

13.2.2.2 The Q-Learning Model

The learning model to be applied to this situation is exactly the same as the one used in the previous situation. The system is still given a particular sensory state, and still chooses a particular action to perform. The consequences of these actions and how they affect its sensory state and rewards are, of course, completely different, since this is a different task being performed. However, this is exactly what the Q-Learning system is supposed to learn, so no changes are needed to the model.

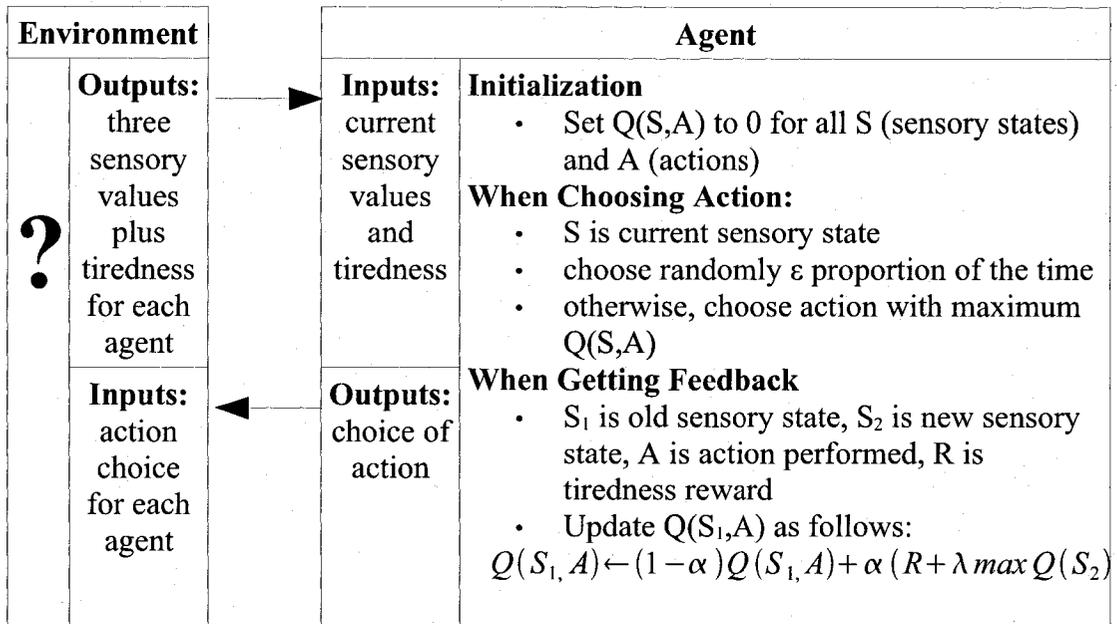


Figure 13.15: Overview of the Q -Learning model for learning to use internal structures for foraging. Note that this is identical to Figure 13.10.

13.2.2.3 Results

As before, the number of foraging trips made by these learning agents can be compared to that of agents foraging without the ability to remember which way they are going. The results of this comparison can be seen in Figure 13.16. As can be seen, there is an advantage to having this sort of internal memory system, and the agents were able to learn to use it to improve their foraging rate. Also, as before, there was an initial disadvantage to having this ability.

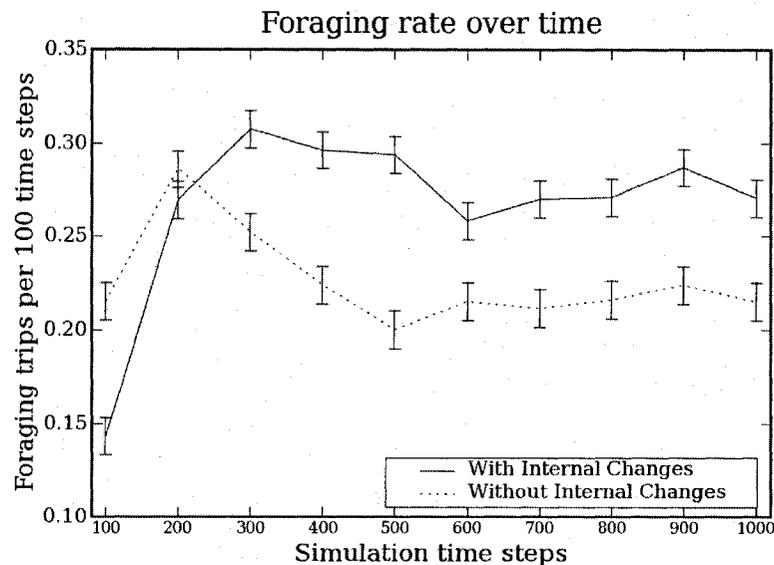


Figure 13.16: Foraging rate over time with and without internal structure formation.

However, it is notable that there is a much smaller advantage to having this internal network, as compared to the large advantage seen for the external epistemic structures. Given this, it may be useful to examine how this advantage changes over different parameter values of the model. This can be done using the same techniques as discussed earlier in this thesis: evaluate the model at different parameter settings and compare these values to some particular standard (i.e. the performance of a model without internal structure generation). However, in this case the question is not whether the model is *equivalent*, but rather is about whether the model is *better* than the standard. It is thus not appropriate to use the equivalence measure that has been the basis of all other analysis in this thesis.

The following graphs show the effects of varying the structure of the internal memory network (the number of hidden nodes and the learning rate) and the parameters for the Q-Learning system. The black lines indicate the 95% confidence interval for the model

without structure generation. All points drawn in a lighter shade indicate parameter settings for which structure generation results in more trips to the target and back (i.e. better performance). These results are representative for other combinations of parameter settings.

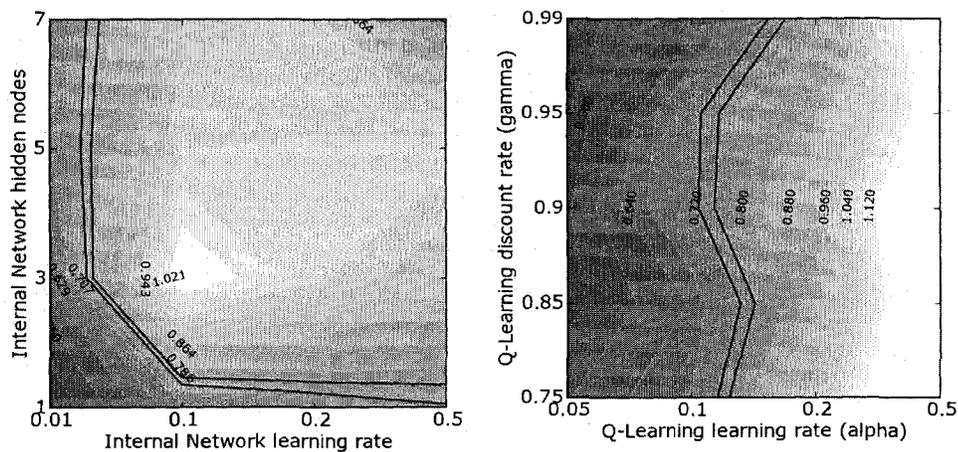


Figure 13.17: Behaviour of internal structure learning model over various parameter settings.

These results indicate that the effect is very robust, and appears for a wide variety of parameter settings. The only requirements are having a sufficient number of hidden nodes in the network (more than 1), a learning rate above 0.01, and a Q-Learning learning rate above 0.1. In these situations, the ability to form internal structures (i.e. to learn to remember particular information in particular contexts) can be made use of in exactly the same way as the external structures. This does not require a complex cognitive mechanism, and can be seen as an early precursor to full representation.

13.3 Summary

13.3.1 Methodology

This chapter demonstrated a very different use of modelling than has been seen in the rest of this thesis. Here, the comparison between model and reality is *qualitative*, not quantitative. Instead of looking for a numerical match between model and reality, the goal is to find models which exhibit a similar pattern of behaviour as the real systems. This can be as simple as determining whether having a particular ability helps a model perform a task better, or as complex as looking at what internal changes to a model result in what sort of high-level behavioural changes.

In the first section, two existing models of peer group interaction were modified to introduce individual differences. The resulting behavioural changes were qualitatively similar to those observed in real subjects. Furthermore, the models both made particular novel predictions about smaller-sized effects. This demonstrates how a model that has been validated via the methodology presented in this thesis can then be used in a more exploratory manner. This exploration can lead to further experimentation to determine if these smaller effects do, in fact, exist in the real subjects, and eventual quantitative modelling of the phenomenon.

In the second section, a completely new model was presented. This model was based on the theory that a simple learning mechanism could be used to allow organisms to make use of structures to simplify long-term tasks. The computational model of this was not meant to match some particular situation, but was meant to be an abstract example of these sorts of learning situations. The success of the model (i.e. the fact that it was able

to learn to use the structures to perform better than agents without the ability to create structures) acts as an existence proof that this sort of process is a possible cognitive model of this phenomenon. This can lead to further research to eventually develop a more concrete and quantitative model.

13.3.2 Contributions

- It was shown that both the mathematical model and the ACT-R model of peer group interactions respond to individual differences in qualitatively similar manners. Effects such as the Hostile Attribution Bias and the facts that Popular children tend to have better social skills, Neglected children interact less often, and Controversial children exhibit a wider range of behaviours are all seen in both models. This further validates them as good models of this phenomenon, even though there are no precise quantitative measures to which the model can be matched using the equivalence method.
- Both the mathematical model and the ACT-R model of peer group interactions were found to exhibit a few surprising phenomena. Controversial children tended to adapt more quickly and interact more often than others. Also, all of the traits strongly associated with being Rejected also tended to be seen in the Neglected children, but to a lesser degree. These findings have only been seen in the model; future work would be needed to establish their presence in real children.
- A qualitative model of epistemic structure formation was investigated, demonstrating that it is possible for a simple learning mechanism to form

structures in the environment that help it in the long term. This can occur without long-term planning, and without requiring a genetic component to the learning (i.e. all learning can take place within the lifetime of the individual).

Furthermore, exactly the same learning algorithm, if used in a situation where it can chose to perform actions that affect an *internal* memory, can learn to use this internal memory in the same way it learned to use the external structures. This could be a possible precursor to generic representation.

14 Conclusions

The subject matter of cognitive science is highly complex. This has led to the development of more and more complicated theories to account for the cognitive phenomena that have been observed. Given these complexities, it is difficult to reliably determine what the implications are of these theories. This is especially true if quantitative predictions are desired. To address this problem, the theories are often converted into computational cognitive models. These models are then used to determine the implications of these theories by observing the behaviour that is produced when these models are put into different experimental situations.

If these computational models are meant to explain particular cognitive processes found in particular cognitive systems, then there must be some correspondence between the model and reality. As discussed in chapter 3, there are different forms of equivalence that can be considered. Behavioural equivalence is the most straightforward, as it simply specifies that the responses by the model in a given situation should be the same as the responses by the real system in that same situation. Of course, presenting this same situation to the model involves creating a model of the environment as well.

Algorithmic equivalence (also known as strong equivalence) is harder to establish. Here, the internal steps of the model should be the same as those of the real system. Since these steps cannot be directly observed, there must be some indirect way of establishing this similarity. To do this, the model must be constrained in terms of how it handles *time*. That is, aspects of behaviour like reaction times cannot be treated as simple outputs of the

model. Instead, the time course of the model must be based on the internal operations occurring within the model. A model which produces accurate time predictions in this manner over a variety of situations (while still achieving behavioural equivalence) exhibits this desirable strong equivalence, and should be an important goal of current modelling research..

Furthermore, the constraint that models should eventually be shown to be structurally equivalent to the real world indicates that each component of the model must be physically possible. This means that models which are meant to explain cognitive phenomena should be constrained to be process models, and thus only deal with the data available at a particular point in time.

Other than these constraints on how a model must be organized, the main consideration when examining a model is that it have particular numerically measured behaviours which match with the numerically measured behaviours of the real system. The difficulties in establishing such a match were discussed in chapter 4. The main conclusion from this work was that the standard statistical measures used by cognitive scientists are limited in terms of what conclusions they can draw about a given model. These measures usually do not take into account the sampling error (for either the real-world data or the model data), and they usually average over a large number of measures. This leads to results such as “this model has an average error of 12% over this set of fifty different measures as compared to the average behaviour of the subjects sampled”. This conclusion is prone to over-fitting due to sampling error, and does not indicate what measures the model is better or worse at modelling.

To address this measurement problem, the relativized equivalence measure was introduced. This is an original measure which is meant to indicate an upper bound on how *wrong* the model may be. Given a particular level of confidence (usually 95%), this measure indicates what the maximum difference could be between the model behaviour and the real behaviour. This can be applied to any measure, and makes no assumptions about the underlying distribution of the data. Furthermore, it can be applied to any statistic, and so can evaluate how different the *variance* of the model data and real data are, or the median, mode, skew, kurtosis, and so on. This is important since a perfect model's behaviour should be statistically indistinguishable from the real data in all respects, not just that the mean behaviour should be the same. The equivalence measure also allows modellers to specify exactly what counts as a “good enough” model. That is, each measure can have a particular range of accuracy that is desired. This allows for a gradual increase in the accuracy of models, as theories develop more completely.

This concept of measuring equivalence allows for a new sort of conclusion from computational cognitive modelling research. Instead of indicating the average difference between the model and some set of sample data, conservative bounds can be placed on the accuracy of the model. Conclusions can indicate “this model has an *worst-case* error of 20% across *all* of the measures considered”. This conclusion is one that takes into account the difference between the sample statistic and the population statistic, and so refers to the relationship between the model and the whole population that it is meant to be modelling, rather than just the few individuals who were part of the study being compared to. This is a more accurate and reliable conclusion than is allowable using

standard statistical techniques.

14.1 Modelling Methodology

All of these considerations, plus the discussion in chapter 5 about the importance of being able to communicate the full details of models to other researchers, combine together into the modelling methodology presented in chapter 6. This details the process of defining the model of the experimental situation, defining the model(s) themselves, analyzing the models, and communicating the results of this research.

First, the specification of the environmental situation requires explicit mention of how the real-world stimuli, behaviours, and measures are adapted for the simulation. Second, computational models are implemented based on the theory being tested. Extremely simplistic models can be implemented first, both as a way of testing the implementation and to serve as a benchmark for comparison for more complex models. Third, the models are evaluated using the equivalence measure. This must be done for a wide range of parameter settings, as the model's behaviour is generally highly dependent on these settings. Indeed, each setting can be thought of as specifying a *different*, but related model. Furthermore, non-numerical parameters can be adjusted which change the structure of the model itself. These results can be communicated completely only by complete publication of all source code, including the data used for comparison and the models of the experimental situations.

This modelling methodology is also an iterative process. If no models are equivalent on the set of measures under consideration, then earlier parts of the methodology must be

revisited. This can involve simply removing particular measures from consideration, expanding the set of parameters searched, or the creation of whole new models. If too many (or all) models are equivalent, then this can be an indication that more measures should be added. If so, then it is preferable that these new measures be of different types of phenomena than the original measures, so as to indicate the generalization of the model to new situations.

14.2 Modelling Toolkit

To facilitate the use of this methodology, a set of software tools has been developed. This toolkit, the Carleton Cognitive Modelling Suite, is the basis of all of the computational modelling work done in this thesis. It allows for the creation of models of both cognitive systems and the environments around them. It also enforces timing constraints, ensuring models' time predictions are related to their internal algorithms. It is integrated with software for running large sets of simulations with different parameter settings, collecting together the resulting data, performing equivalence measures on the data, and generating the graphs and analysis seen in this thesis.

This toolkit is not the only way to follow the methodology described in this thesis. Indeed, it has not even been established that this toolkit is usable by other researchers for tasks other than those seen in this thesis. However, the results obtained from following this methodology were made possible by this toolkit, and led to conclusions not possible using standard analysis tools. It is hoped that future work can further develop this toolkit and establish it as one that is useful to the wider cognitive science community.

The toolkit also has integrated with it the core components of the ACT-R cognitive modelling architecture. This includes a procedural memory system and a declarative memory system, as well as a standard mechanism for communication between modules. These components have been extensively used by the ACT-R modelling community to model various aspects of human cognition. Given these components, it is possible to integrate them with other models, or use them to quickly create ACT-R based models that can be evaluated with the CCMSuite. This allows for direct comparison of ACT-R based models with other modelling approaches, since the exact same environmental model can be used.

14.3 Modelling Results

Applying this methodology has led to a collection of novel cognitive modelling results. In each case, the equivalence measure allows for a different kind of conclusion than is normally made. This more accurately describes the capabilities of the models, and clearly indicates the aspects of the cognitive phenomena that are successfully captured by the model.

One set of results addresses the Repeated Binary Choice task. A variety of models were examined, including the RELACS model developed by Erev and Barron (2005), and models based on the ACT-R declarative and procedural memory systems. One core finding is that the RELACS model significantly outperforms the ACT-R models. This comparison with a different modelling approach was not done in the original RELACS research.

More importantly, RELACS was found to *not* be a good generic model of the Repeated Binary Choice task. Instead, it only provides equivalent results in situations where there is a simple distribution of rewards. That is, the rewards must be something like “pressing A gives a reward of 5 half the time and 2 the other half of the time”, rather than rewards determined by Gaussian distributions. Furthermore, the model does not handle situations which differ only by scaling the amount of the reward, or situations where the rewards for both choices are a fixed value. This is in contrast to the original conclusions about the model, where all of these different conditions were averaged together.

One further question addressed by this work is whether the system for choosing between strategies in RELACS is necessary, or whether it could be replaced by a random choice mechanism. In the original work, it was found that a random choice system has the same average error as the standard RELACS system. However, in this thesis, it was determined that switching to a random choice system leads to models which are not equivalent. This is an encouraging result, especially since RELACS is named for the aspect of the system which allows for Reinforcement Learning Among Cognitive Strategies.

Another set of modelling results in this thesis involve interactions within peer groups. Here, a group of individuals adjust how much they like each other based on their interactions. A standard sociometric measurement (CDC) is then used to classify individuals as Popular, Rejected, Neglected, Controversial, or Average. Both a mathematical model and an ACT-R model were created and evaluated, and both were able to capture the classification patterns and the stability of those classifications.

Furthermore, exploratory investigations indicate an underlying set of behavioural patterns than may match individual differences in real people.

Finally, the framework was applied to a situation where no quantitative comparison could be made. Here, a standard learning algorithm was used in a novel context, where it was possible to create structures in the environment that could simplify a foraging task. This same learning algorithm was also able to learn to create internal structures to allow agents to remember particular aspects of their task. By examining a wide range of parameter settings, it was established that this capability for learning is a robust phenomenon (i.e. one that appears over a broad range of parameter settings, and does not rely on specific parameter tuning for a specific situation). This work serves as an existence proof that this learning situation is a possible candidate for simple cognitive systems to use their environment and their own memory capacities in novel ways.

All of these findings are useful and new cognitive science results, and are part of ongoing research programs. Despite the large differences between these models and their applications, they all fit within the computational cognitive modelling methodology described in this thesis. By using this methodology, these findings are more rigorous and reliable than is normally found in modelling research. This approach is thus needed for computational modelling to be a core part of scientific research into cognitive systems.

15 References

- Anderson, J. R., Albert, M. V., & Fincham, J.M. (2005). Tracing problem solving in real time: fMRI analysis of the subject-paced Tower of Hanoi. *Journal of Cognitive Neuroscience*, *17*, 1261-1274.
- Anderson, J. R., Qin, Y., Stenger, V. A., & Carter, C. S. (2004). The relationship of three cortical regions to an information-processing model. *Journal of Cognitive Neuroscience*, *16*(4), 637-653.
- Anderson, J. R., Qin, Y., Sohn, M-H., Stenger, V. A. & Carter, C. S. (2003). An information-processing model of the BOLD response in symbol manipulation tasks. *Psychonomic Bulletin & Review*, *10*, 241-261.
- Anderson, J. R. & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Erlbaum.
- Anderson, M. (2006). Evidence for massive redeployment of brain areas in cognitive functions. *Neuroscientist*, *13*, 13-21.
- Axelrod, R. (1981). The evolution of cooperation. *Science*, *211*, 1390-6.
- Axelrod, R. (1997). Advancing the art of simulation in the social sciences. In R. Conte, R. Hegselmann, & P. Terna (Eds.), *Simulating Social Phenomena*. Berlin: Springer.
- Axtell, R., Axelrod, R., Epstein, J.R., & Cohen, M.D. (1996). Aligning simulation models: A case of study and results. *Computational Mathematical Organization Theory*, *1*(2), 123-141.
- Barker L. E., Luman E.T., McCauley M.M., & Chu Y.R. (2002). Assessing equivalence: An alternative to use of difference tests for measuring disparities in vaccination coverage. *American Journal of Epidemiology*, *156*, 1056-1061.
- Beaulieu-Prévost, D. (2006). Statistical decision and falsification in science: going beyond the null hypothesis. In Hardy-Vallée, B., (Ed.), *Cognitive Decision-Making: Empirical and Foundational Issues*. Cambridge:Cambridge Scholars Press.
- Bechtel, W. & Richardson, R.C. (1993). *Discovering complexity: Decomposition and localization as strategies in scientific research*. Princeton: Princeton University Press.
- Bechtel, W. (2005). *Mental mechanisms: What are the operations?* Proceedings of the 27th Annual Meeting of the Cognitive Science Society.
- Bonabeau E., Dorigo M., & Theraulaz G. (1999). *Swarm intelligence: From natural to artificial systems*. Santa Fe Institute studies in the sciences of complexity. New York:Oxford University Press.
- Bradbury, J.W., & Vehrencamp, S.L. (1998). *Principles of animal communication*.

- Sunderland, Mass: Sinauer Associates.
- Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. MIT Press.
- Breiman, L., Friedman, J. H., Olsen, R., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth, Belmont, California.
- Buller, David J. (1993). Confirmation and the computational paradigm (or: Why do you think they call it artificial intelligence?). *Minds and Machines*, 3, 155-181.
- Busemeyer, J. R. & Yi-Min Wang (2000). Model comparisons and model selections based on generalization test methodology. *Journal of Mathematical Psychology*, 44, 171-189.
- Carley, K. M. (1996). *Validating computational models*. Social and Decision Sciences Working Paper. Pittsburgh, PA: Carnegie Mellon University.
- Carpenter, G.A., Grossberg, S., Markuzon, N., Reynolds, J.H., & Rosen, D.B. (1992), Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3, 698-713.
- Chandrasekharan, S. (2005). *Epistemic structure: An inquiry into how Agents change the world for cognitive congeniality*. PhD Thesis, Institute of Cognitive Science, Carleton University, Ottawa, Canada.
- Chandrasekharan, S. & Stewart, T.C. (2007) . The origin of epistemic structures and proto-representations. *Adaptive Behaviour*. 15(3), 329-359.
- Cillessen, A., & Bukowski, W. (2000). Conceptualizing and measuring peer acceptance and rejection. *New Directions for Child and Adolescent Development*, 88, 3-10.
- Cillessen, A., Bukowski, W., & Haselager, G. (2000). Stability of sociometric categories. *New Directions for Child and Adolescent Development*, 88, 75-93.
- Clark A. (1997). *Being there: Putting brain, body, and world together again*. Cambridge MA: MIT Press
- Clark, R. E. & Sugrue, B., (1989). Research on instructional media, 1978-1988. In Ely, D. (Ed.) *Educational Media Yearbook 1988-89*. Littleton, Co.: Libraries Unlimited.
- Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). New Jersey: Lawrence Erlbaum.
- Cohen, J. (1994). The earth is round ($p < .05$), *American Psychologist*, 49(12), 997-1003.
- Coie, J.D. & Dodge, K.A., (1988). Multiple sources of data on social behavior and social status. *Child Development*, 59, 815-829.
- Coie, J.D., Dodge, K.A., & Coppotelli, H. (1982). Dimensions and Types of Social Status: A Cross-Age Perspective. *Developmental Psychology*, 18, 557-570.
- Coie, J.D., Dodge, K.A., & Kupersmidt, J.B. (1990). Peer group behavior and social

- status. In S.R. Asher & J.D. Coie (Eds.), *Peer rejection in childhood*. New York: Cambridge University Press.
- Cole, M. & Means, B. (1981). *Comparative studies of how people think: An introduction*. Cambridge, MA: Harvard University Press.
- Connell, L. & Keane, M. (2006). A model of plausibility. *Cognitive Science*, 30(1), 95-120.
- Corbett, A. T., Anderson J. R., & O'Brien, A. T. (1993). The predictive validity of student modeling in the ACT Programming Tutor. In P. Brna, S. Ohlsson, & H. Pain (Eds.) *Artificially Intelligence and Education, 1993: The Proceedings of AI-ED 93*. Charlottesville, VA: AACe.
- Crick, N. R., & Dodge, K. A. (1994). A review and reformulation of social information-processing mechanisms in children's social adjustment. *Psychological Bulletin*, 115, 74-101.
- Davison, A.C. & Hinkley, D.V. (1997). *Bootstrap methods and their application*. Cambridge University Press.
- Denison, D. (1997). *Simulation based Bayesian nonparametric regression methods*. PhD Thesis, Imperial College, London.
- Dennett, D. (1975). Why the law of effect will not go away. *Journal of the Theory of Social Behaviour*, 5, 179-87.
- Dennett, D. (1978) Why you can't make a computer that feels pain. *Synthese*, 38(3), 415-456.
- Dennett, D. (1991). Real patterns. *Journal of Philosophy*, 88, 27-51.
- Dodge, K.A., Coie, J.D., & Brakke, N.P. (1982). Behavioral patterns of socially rejected and neglected preadolescents: The roles of social approach and aggression. *Journal of Abnormal Child Psychology*, 10, 389-409.
- Dodge, K. A., Lansford, J., Burks, V., Bates, J.E., Pettit, G., Fontaine, R., & Price J.M. (2003). Peer rejection and social information-processing factors in the development of aggressive behavior problems in children. *Child Development*, 74, 374-393.
- Dretske F. (1994) If you can't make one, you don't know how it works. In P. French, T. Uehling, & H. Wettstein (Eds.) *Midwest Studies in Philosophy*, 19, 468-82.
- Einstein, A. & Infeld, L. (1938). *The Evolution of Physics*. New York: Simon and Schuster.
- Eliasmith, C. & C. H. Anderson (2003). *Neural Engineering: Computation, representation and dynamics in neurobiological systems*. MIT Press.
- Elman, J.L. (1990). Finding structure in time. *Cognitive Science*, 14, 179-211.
- Epp, E. (1993). *The analog/digital distinction in the philosophy of mind*. M.A. Thesis, Simon Fraser University.

- Erev, I. & Barron, G. (2005). On adaptation, maximization, and reinforcement learning among cognitive strategies. *Psychological Review*, 112(4), 913-931.
- Erev, I., Bereby-Meyer, Y., & Roth, A.E. (1999), The effect of adding a constant to all payoffs: Experimental investigation, and a reinforcement learning model with self-adjusting speed of learning. *Journal of Economic Behavior and Organizations*, 39(1), 111-128.
- Feynman, R. (1979). *Photons: Corpuscles of light*. The Douglas Robb Memorial Lectures.
- Forster, M. (2000). Key concepts in model selection: performance and generalizability. *Journal of Mathematical Psychology*, 44, 205-231.
- Friedman, M. P., Burke, C. J., Cole, M., Keller, L., Millward, R. B., & Estes, W. K. (1964). Two-choice behavior under extended training with shifting probabilities of reinforcement. In R. C. Atkinson (Ed.), *Studies in mathematical psychology*. 250-316. Stanford, CA: Stanford University Press.
- Giere, R. (1988). *Explaining Science: A Cognitive Approach*. Chicago: University of Chicago Press.
- Giere, R. (1999). *Science Without Laws*. Chicago: University of Chicago Press.
- Golden, R. (2000). Statistical tests for comparing possibly misspecified and nonnested models. *Journal of Mathematical Psychology*, 44, 153-170.
- Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11, 23-63.
- Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42, 335-346.
- Henry, J.D. (1977). The use of urine marking in the scavenging behaviour of the red fox (*Vulpes vulpes*). *Behaviour*, 62, 82-105.
- Hersh, R. (1997). *What is mathematics, really?* Oxford University Press.
- Hills, T. (2006) Animal foraging and the evolution of goal-directed cognition. *Cognitive Science*, 30, 3-41.
- Holland, J.H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, University of Michigan Press.
- Humphrys, M. & O'Leary, C. (2002). *Constructing complex minds through multiple authors*. Proceedings of the 7th International Conference on Simulation of Adaptive Behavior (SAB-02).
- Ioannidis J.P.A. (2005) Why most published research findings are false. *Public Library of Science Medicine* 2(8), 124.
- Kohonen, T. (1995). *Self-organizing maps*. Series in Information Sciences, 30. Heidelberg: Springer.
- Koza, J. (1992). *Genetic programming: On the programming of computers by means of*

natural selection. MIT Press.

- Lane, P.C.R. & Gobet, F. (2003). Developing reproducible and comprehensible computational models. *Artificial Intelligence*, 144, 251-263.
- Lebiere, C. & Wallach, D. (2001). Sequence learning in the ACT-R cognitive architecture: Empirical analysis of a hybrid model. In R. Sun & C. L. Gilles (Eds.). *Sequence Learning: Paradigms, Algorithms, and Applications*. Berlin: Springer Lecture Notes in Computer Science.
- Lebiere, C., & West, R. L. (1999). *A dynamic ACT-R model of simple games*. In Proceedings of the Twenty-first Conference of the Cognitive Science Society, 296-301. Mahwah, NJ: Erlbaum.
- Moreno, J. L. (1934). *Who shall survive? A new approach to the problem of human interrelations*. Washington, D. C.: Nervous and Mental Disease Publishing Co.
- 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*, 66, 521-522.
- Myung, I. J., & Pitt, M. A. (2002). Mathematical modeling. In J. Wixten (ed.), *Stevens' Handbook of Experimental Psychology* (3rd Edition), 429-459. New York, NY: John Wiley & Sons.
- Newcomb, A.F., & Bukowski, W.M. (1983). Social impact and social preference as determinants of children's peer group status. *Developmental Psychology*, 19, 856-867.
- Newcomb, A.F., Bukowski, W.M., & Pattee, L. (1993). Children's peer relations: A meta-analytic review of popular, rejected, neglected, controversial, and average sociometric status. *Psychological Bulletin*, 113, 99-128.
- O'Reilly, R. & Munakata, Y. (2000). *Computational Exploration in Cognitive Neuroscience*. MIT Press.
- Parkhurst, J.T., & Hopmeyer, A. (1998). Sociometric popularity and peer-perceived popularity: Two distinct dimensions of peer status. *Journal of Early Adolescence*, 18, 125-144.
- Popper, K. (1959). *The Logic of Scientific Discovery*. New York: Basic Books.
- Pyke, P., West, R.L., & LeFevre, J. (2007). *How Readers Retrieve Referents for Nouns in Real Time: A Memory-based Model of Context Effects on Referent Accessibility*. 8th International Conference on Cognitive Modeling.
- Pylyshyn, Z. (1984) *Computation and cognition*. Cambridge: MIT Press.
- Pylyshyn, Z. (1989). Computing in Cognitive Science. In Posner, M. (Ed.) *Foundations of Cognitive Science*. Cambridge: MIT Press.
- Rechenberg, I. (1994) *Evolutionsstrategie '94*. Stuttgart: Frommann-Holzboog.
- Rescorla, R.A., & Wagner, A.R. (1972) A theory of Pavlovian conditioning: Variations in

- the effectiveness of reinforcement and nonreinforcement. In A.H. Black & W.F. Prokasy (Eds.) *Classical Conditioning II*.
- Roberts, S. & Pashler, H. (2000). How persuasive is a good fit? A comment on theory testing. *Psychological Review*, 107, 358-367.
- Ross, D. (2000). Rainforest realism: A Dennettian theory of existence. In D. Ross, D. Thompson and A. Brook (Eds.) *Dennett's Philosophy: A Comprehensive Assessment*. Cambridge: MIT Press.
- Roth, A. E., & Erev, I. (1995). Learning in extensive-form games: Experimental data and simple dynamic models in intermediate term. *Games and Economic Behavior, Special Issue: Nobel Symposium*, 8, 164-212.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986) Learning representations by back-propagating errors. *Nature*, 323, 533-536.
- Rutledge-Taylor, M. & West, R. L. (2007). *MALTA: Enhancing ACT-R with a holographic persistent knowledge store*. 29th Annual Meeting of the Cognitive Science Society.
- Schmidt, F. (1996). Statistical significance testing and cumulative knowledge in psychology: Implications for the training of researchers. *Psychological Methods*, 1, 115-129.
- Semaphore Corp. (2005) SHRDLU Resurrection.
<<http://www.semaphorecorp.com/misc/shrdlu.html>>
- Shallice, T. & Burgess, P. (1996). The domain of supervisory processes and temporal organization of behaviour. *Philosophical Transactions of the Royal Society of London B*. 351, 1405-1412.
- Sibley, D.E. & Kello, C.T. (2004). Dissociations between regularities and irregularities in language processing: Computational demonstrations without separable processing components. *In the Proceedings of the 26th Annual Meeting of the Cognitive Science Society*.
- Siegel, S. & Goldstein, D. A. (1959). Decision-making behavior in a two-choice uncertain outcome situation. *Journal of Experimental Psychology*, 57, 37-42.
- Silberman, S. (2003). The Bacteria Whisperer. *Wired*, 11.04.
- Stewart, T.C. (2004). *Teaching Computational Modelling to Non-Computer Scientists*. Sixth International Conference on Cognitive Modelling.
- Stewart, T.C. (2006). *Tools and Techniques for Quantitative and Predictive Cognitive Science*. 28th Annual Meeting of the Cognitive Science Society.
- Stewart, T.C. & West, R.L. (2007). Deconstructing and reconstructing ACT-R: Exploring the architectural space. *Cognitive Systems Research*, 8(3), 227-236.
- Stewart, T.C., West, R.L., & Coplan, R. (2007). Multi-agent models of social dynamics in children. *Cognitive Systems Research*, 8(1), 1-14.

- Stopka, P. & Macdonald, D. W. (2003) Way-marking behavior: an aid to spatial navigation in the wood mouse (*Apodemus sylvaticus*). *BMC Ecology*, 3(3).
- Sutton, R & Barto, A. (1998) *Reinforcement learning: An introduction*. Cambridge: MIT Press.
- Thagard, P. (1995). *Mental leaps*. Cambridge: MIT Press.
- Tryon, W. (2001). Evaluating statistical difference, equivalence, and indeterminacy using inferential confidence intervals: An integrated alternative method of conducting null hypothesis statistical tests. *Psychological Methods*, 6(4), 371-386.
- Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27.
- West, R., Stewart, T.C., Lebiere, C., & Chandrasekharan, S. (2005). *Stochastic resonance in human cognition: ACT-R versus game theory, associative neural networks, recursive neural networks, Q-Learning, and humans*. 27th Annual Meeting of the Cognitive Science Society.
- Wilkinson, L., & Task Force on Statistical Inference. (1999). Statistical methods in psychology journals: Guidelines and explanations. *American Psychologist*, 54, 594-604.
- Zahavi, A., & Zahavi, A. (1997). *The handicap principle: A missing piece of Darwin's puzzle*. Oxford: Oxford University Press.

16 Appendix A: The Carleton Cognitive Modelling Suite

This appendix is the User's Guide for the Carleton Cognitive Modelling Suite (CCMSuite). This software was developed as part of this thesis to support the methodology developed herein. It is meant to be accessible to both expert and novice programmers. The core idea is to be able to express the models and modelling process as clearly in a manner that is as close as possible to the normal verbal descriptions of the model. Non-programmers should be able to look at a model built with CCMSuite and be able to reasonably understand what is occurring.

The CCMSuite was used for all of the modelling in this thesis. It has also been used as the core of CGSC 5001: Cognition and Artificial Systems (taught in Winter 2004, Fall 2004, and Fall 2005), and CGSC 6501: Carleton Cognitive Modelling Summer Workshop (taught Summer 2006). It is a further development of the modelling software described in (Stewart, 2004). It is freely available as open source software (licensed under the GNU General Public License) at <http://ccmlab.ca/ccmsuite.html>.

These instructions cover installing the software (on Windows, OS X, or Linux), creating environments and models, displaying the outputs and internal progress of the model as it runs, running the models across multiple computers, performing statistical analysis on the results, exploring the effects of different parameter settings, and sharing the results with other researchers.

16.1 Installation

CCMSuite is capable of running on any platform that supports the Python programming

language. It makes use of the TurboGears content management system and the Matplotlib graphing library.

16.1.1 Installing Python

CCMSuite uses the Python programming language. This is a language designed for clarity and correctness, suitable for both rapid prototyping and large-scale systems. The idea is that programs written in Python should be as readable as possible, even to someone who is unfamiliar with the language.

To install Python, follow the instructions for your particular operating system

- Windows: Download and run <http://www.python.org/ftp/python/2.4.3/python-2.4.3.msi>
- Mac OS X: Follow the instructions at <http://www.python.org/download/mac/>
- Linux: Python is probably already installed. You should use whatever package manager comes with your Linux distribution to ensure you have at least Python version 2.4 installed.

16.1.2 Installing CCMSuite

CCMSuite is currently undergoing development. This means that the software is changing on a daily basis. To make sure that you have the very latest version, you can install CCMSuite in such a way as to allow you to quickly download all the latest changes as they occur.

The software we use to do this is called Subversion. This system allows files to be synchronized across different computers. By giving your computer a single command, it can connect to the main CCMSuite website and update every file that has changed since

you last downloaded the system.

The process for doing this is different depending on which operating system your computer uses. Please follow the instructions below that best describe your situation.

- **Windows 2000/XP** (Option 1, for people who prefer graphical user interfaces):
 - Install TortoiseSVN <<http://tortoisesvn.tigris.org/>>
 - Restart your computer
 - Go to <svn://rob.ccmlab.ca/ccmlab/CCMSuite> with your web browser
 - Right-click on CCMSuite
 - Select Checkout...
 - In the box labelled Checkout directory, choose the folder you wish to put the CCMSuite source code into (for example, C:\CCMSuite).
 - Press OK.
 - Your computer will now automatically download the latest version of the software to C:\CCMSuite.
 - Whenever you wish to re-download the system to ensure you have the latest version, go to C:\, right-click on CCMSuite, and select SVN Update.
- **Windows NT/2000/XP** (Option 2, for people who prefer using the command line):
 - Install <http://subversion.tigris.org/files/documents/15/31465/svn-1.3.1-setup.exe>
 - Run `svn checkout svn://rob.ccmlab.ca/ccmlab/CCMSuite C:\CCMSuite`
 - you can change C:\CCMSuite to some other directory if you wish
 - Whenever you wish to re-download the system to ensure you have the latest version, run

```
svn update C:\CCMSuite
```

- **Mac OS X:**

- Install <http://metissian.com/downloads/macosx/subversion/subversion-client-1.3.1.dmg>

- Run the following command in Terminal:

```
/usr/local/bin/svn checkout svn://rob.ccmlab.ca/ccmlab/CCMSuite  
~/CCMSuite
```

- you can change ~/CCMSuite to some other directory if you wish. If you leave it like this, it will create a directory called CCMSuite in your home directory.
- Whenever you wish to re-download the system to ensure you have the latest version, run

```
/usr/local/bin/svn update ~/CCMSuite
```

- **Linux:**

- Subversion is most likely already installed. If not, use whatever package manager your Linux distribution uses to install subversion. This is often done with a command like

```
apt-get install subversion
```

- Run

```
svn checkout svn://rob.ccmlab.ca/ccmlab/CCMSuite ~/CCMSuite
```

- you can change ~/CCMSuite to some other directory if you wish
- Whenever you wish to re-download the system to ensure you have the latest version, run

```
svn update ~/CCMSuite
```

16.1.3 Configuring CCMSuite

Python needs to know where you installed CCMSuite so that it can find it. The easiest way to do this is to just put all of your models in the C:\CCMSuite directory. All programs run from there will automatically find all of the CCMSuite code, and everything will work fine.

If you wish to be able to run programs that use CCMSuite from anywhere, you need to tell Python where to find it. The most general way to do this is to add a small file into the site-packages directory. On windows, this will be in some location like

```
C:\Python24\Lib\site-packages
```

but this location may vary depending on where it has been installed. A similar directory will be found for OS X and Linux. You should make a file in this directory called CCMSuite.pth. The contents of the file should be the directory you have put CCMSuite into. For example, if you installed it to C:\CCMSuite, as suggested above for windows, you would make a text file (using Notepad or some other text editor) with the following contents:

```
C:\CCMSuite
```

16.2 Creating Models Environments

The first step for creating a model is to create a model of the environment. This defines the experimental situation the model will be put into. This must describe all of the stimuli observed, and what exactly happens when the participant performs different actions. Also, all of the experimental measures must be defined here. As discussed in chapter 2, defining this model of the environment specifies exactly how the real-world stimuli and measurements are mapped onto the model. Importantly, this system is designed before the model of the participant is created. This allows the same environment to be used for many different models, and thus comparisons between models are possible.

16.2.1 Example 1: A Blinking Light

The following program defines an environment where a single light turns on and off ten times.

```
import ccm2 as ccm
log=ccm.log()

class BlinkingLightEnvironment(ccm.Model):
    def start(self):
        for i in range(10):
            self.light='on'
            yield 0.5
            self.light='off'
            yield 0.5
        self.stop()

e=BlinkingLightEnvironment()
ccm.log_everything(e)
e.run()
```

The first two lines are needed when defining any CCMSuite model. The first line tells the system to make use of the most recent version of CCMSuite (ccm2), and the second line enables the logging system so we can see what is occurring inside the model.

```
import ccm2 as ccm
log=ccm.log()
```

Next, the actual environment is defined. The first line specifies what we are going to call the model (in this case, it is called `BlinkingLightEnvironment`).

```
class BlinkingLightEnvironment(ccm.Model):
```

Next, we define what the model does. Every model can define a “start” function which gives instructions for what should happen when the model starts running. In this case, we use a `for` statement to repeat 10 times a sequence of commands. Those commands are first to turn a light on (`self.light='on'`), then wait for 0.5 seconds (`yield 0.5`), then turn the light off, and wait another 0.5 seconds. After this is repeated 10 times, the command

`self.stop()` tells the simulation to stop running.

```
def start(self):
    for i in range(10):
        self.light='on'
        yield 0.5
        self.light='off'
        yield 0.5
    self.stop()
```

repeat 10 times
turn light on
wait 0.5 seconds
turn light off
wait 0.5 seconds
stop the simulation

Once the environment has been defined, we now need to run it. This is accomplished with the following commands

```
e=BlinkingLightEnvironment()
ccm.log_everything(e)
e.run()
```

The first line actually creates the environment that was defined. The second line is a special command which tells the system to automatically display everything that goes on inside the model. The final line actually runs the simulation.

To run the program, either enter the command “python filename.py” (replacing filename with the name of the file) or, in Windows, open the file with IDLE and choose Run->Run Module from the menu. The output is as follows, with the first number in each line indicating the time at which it occurs.

```
0.000 light on
0.500 light off
1.000 light on
1.500 light off
2.000 light on
2.500 light off
3.000 light on
3.500 light off
4.000 light on
4.500 light off
5.000 light on
5.500 light off
6.000 light on
```

```

6.500 light off
7.000 light on
7.500 light off
8.000 light on
8.500 light off
9.000 light on
9.500 light off

```

16.2.2 Example 2: Responding to the Participant

In order for environments to be useful, they must be able to respond to actions by the participant, and they must be able to provide stimuli to the participant. In this example, the blinking light environment will be changed into a reaction time test. Here, when the light turns on, the participant must press a button as quickly as possible. For now, we will not worry about measuring how quickly they respond (this will be addressed in the next section). Instead, we will just give the participant a score based on the number of times they do press the button.

```

class ReactionTimeEnvironment(ccm.Model):
    def start(self):
        self.score=0
        for i in range(10):
            self.light='on'
            yield self.press
            self.light='off'
            yield 0.5
        self.stop()

    def press(self):
        if self.light=='on':
            self.score=self.score+1

```

The first change is that at the beginning of the start function, we set the current score to be zero. This value will be increased whenever a button is pressed.

```
self.score=0
```

The second change is the addition of a press() command. The model of the participant will use this to press the button. All it does is first check to see if the light is on, since we

don't want anything to happen if the participant presses the button when the light is off.

If the light is on, then the score is increased by one.

```
def press(self):
    if self.light=='on':
        self.score=self.score+1
```

The third and final change is that the light no longer stays on for 0.5 seconds. Instead of a “yield 0.5” command, we now have a “yield self.press” command. While the old one means “wait for 0.5 seconds”, the new command tells the system to wait until the press command occurs. In other words, instead of waiting for 0.5 seconds and then turning the light off, the light will be turned off when the button is pressed.

```
yield self.press
```

To run this model, we need to create both the environment and a model of the participant.

The model of the participant is created in *exactly* the same way as the model of the environment. Of course, the participant performs very different actions, but the underlying capabilities are the same. Anything that can be done in the environment can be done in the model of the participant.

In this case, we want a simple participant that watches for the light to be on and then presses the button. However, in order to add a little bit of variability to the model, it is set up to wait for random periods of time between checking to see if the light is on.

```
class RandomPressModel(ccm.Model):
    def start(self):
        while True:
            if self.env.light=='on':
                self.env.press()
            yield self.random.uniform(0.1,0.3)
```

The first two lines should be familiar from the previous examples. The “while True” statement causes it to repeat what comes after it forever (or until the simulation finishes).

The next line checks to see if the light is on. The “self.env.light” statement is how we refer to the light in the environment. If that light is on, then the it presses the button. It then waits for a random period of time between 0.1 seconds and 0.3 seconds before checking again.

In order to run this model, we need to create both the environment and the participant.

This is done the same way as before. The one addition is that we need to put the participant in the environment

```
e=ReactionTimeEnvironment()  
ccm.log_everything(e)  
e.model=RandomPressModel()  
e.run()
```

Here is the result of running this simulation.

```
0.000 score 0  
0.000 light on  
0.000 score 1  
0.000 light off  
0.500 light on  
0.684 score 2  
0.684 light off  
1.184 light on  
1.429 score 3  
1.429 light off  
1.929 light on  
1.956 score 4  
1.956 light off  
2.456 light on  
2.591 score 5  
2.591 light off  
3.091 light on  
3.163 score 6  
3.163 light off  
3.663 light on  
3.671 score 7  
3.671 light off
```

```
4.171 light on
4.226 score 8
4.226 light off
4.726 light on
4.785 score 9
4.785 light off
5.285 light on
5.332 score 10
5.332 light off
```

16.2.3 Example 3: Making Measurements

In addition to presenting stimuli and responding the actions, the environment must also specify exactly how the measurements are made. These measurements are vital, since these values are what will be compared between the model and reality. The measurements should be defined so that they are as similar as possible to those in the real world.

In this case, we will add a measurement of the reaction time to the model. What we want to do is measure the amount of time that passes between the light turning on and the button being pressed. Of course, this should be the time *in the simulation*, not the actual amount of computer processing time required. CCMSuite automatically makes this sort of information available based on the time requirements for the individual steps inside the models.

```
def start(self):
    self.score=0
    for i in range(10):
        self.light='on'
        time1=self.now()
        yield self.press
        time2=self.now()
        log.rt=time2-time1
        self.light='off'
        yield 0.5
    self.stop()
```

The trick here is to find out at what time the light went on and at what time the button is

pressed. We do that in the above modified code using the “self.now()” command, which always gives the current time. In this case, time1 is the time when the light turns on, and time2 is the time after the button is pressed. By subtracting the two we get the reaction time. This is stored in the log using the command “log.rt=time2-time1”. This log command allows us to record any measurements desired.

Since we no longer need all of the information about exactly when the light is turned on or off, these can be removed by eliminating the `ccm.log_everything` line. Alternatively, this line can be left in so that all of the details are shown. This can, however, lead to a large volume of data for complex models.

The resulting model gives the following output. This shows the reaction time for each of the 10 button presses.

```
0.000 rt 0.0
0.522 rt 0.0222869409733
1.097 rt 0.0743347692363
1.781 rt 0.1841652759
2.393 rt 0.112050603666
2.897 rt 0.00392343528695
3.543 rt 0.1462386631
4.054 rt 0.0114893917876
4.652 rt 0.0975345336689
5.286 rt 0.133538149055
```

16.3 Visualizing Model Runs

In the previous section, the results of the simulations were shown in text format, with each row showing a time and what occurred at that time. While this is a convenient format for simple models, it becomes significantly less readable as the models get more complicated. For this reason, CCMSuite also provides a table-based output format which

shows the same information but in a more readable and colour-coded manner.

If the command “--html” is added when running the model, then an HTML file will be created. For example, if the model is in a file called “MyModel.py”, then running the command “python MyModel.py --html” will produce a new HTML file called “MyModel.py-log.html”. This file can be opened in any standard web browser.

For the model developed in section 16.2.3, above, the following data might appear. Note that the time values will be different each time it is run, since there is some randomness in the model.

time	light	rt	score
0.467	off		0
0.500	on		
1.084	off	0.126860581119	1
1.127	on		
1.654	off	0.0675888855397	2
1.694	on		
2.178	off	0.0321750700274	3
2.227	on		
2.734	off	0.0481488879589	4
2.775	on		
3.409	off	0.174970473434	5
3.450	on		
3.947	off	0.0474664464086	6
3.997	on		
4.558	off	0.110761640796	7
4.608	on		
5.002	off	0.0246074619762	8
5.133	on		
5.560	off	0.0300853610033	9
5.663	on		
5.666	off	0.00320778564509	10

As a more complex example, here is the display of a Python ACT-R model doing a serial recall task. This view allows one to visually determine what the various different ACT-R components are doing at different times.

time	m					x	
	firing	focus	memory		retrieval		selected
		chunk	busy	error	chunk		
0.000		start list				recall_first_group	
0.050	recall_first_group	group first list	True	False			
0.614			False		name:group1 parent:list position:first	start_recall_of_group	
0.664	start_recall_of_group	item gpos:first	True	False			
1.936		groupname:group1 list:list pos:first	False		name:1 parent:group1 position:first	harvest_first_item	
1.986	harvest_first_item		True	False		1	
2.962			False		name:2 parent:group1 position:second	harvest_second_item	
3.012	harvest_second_item		True	False		2	
3.840			False		name:3 parent:group1 position:third	harvest_third_item	
3.890	harvest_third_item		True	False		3	
5.539			False	True		second_group	
5.589	second_group	group second list	True	False			
6.240			False		name:group2 parent:list	start_recall_of_group	

16.4 Distributed Execution

The preceding sections demonstrated how CCMSuite can be used to create a model, run it, and display what is happening internally while the model is running. However, for the majority of modelling situations, we want to be able to run the model many times and

gather all the resulting data. For now, we will only be considering the case where all of the parameter values are kept the same when running the multiple simulations. Adjusting parameters will be discussed in section 17.6. However, since most cognitive models will have some random aspects, even when parameters are not varied there will be some performance differences. This can be thought of in the same way as running multiple human participants in an empirical study; even if the stimuli are the same, the behaviours will differ. Running the simulation multiple times allows us to measure the distribution of results.

To run these simulations as quickly as possible, CCMSuite allows them to be run on multiple computers. Any computer that has CCMSuite installed can be harnessed to run simulations, and are referred to as *clients*. These will be controlled by a central *server* program that runs on one particular computer. All communication is done via TCP/IP, so all computers need to be able to contact each other over the Internet.

16.4.1 Setting up the Server

The central server program is controlled by a web-based interface, meaning that simulations can be run and data analyzed from anywhere that is Internet-accessible.

However, in order for this to work, the server program must first be run. The first step is to install the extra Python modules that the server needs. This can be done by running the following program from the main CCMSuite directory.

```
python install-extras.py
```

This will connect to the Internet and automatically download the necessary Python

modules. It should work on Windows, OS X, and Linux.

Once this has finished downloading, you can run the server. This requires running two separate programs, and should be done in this order:

```
python start-server.py
python start-ccm.py
```

16.4.2 Adding a Model to the Server

In order for the server to run a model, it must first find out about the model. This is done simply by telling the model to send all of its data to the server.

There are two ways to do this. First, you can modify the `log=ccm.log()` line in the model to look like this:

```
log=ccm.log(trace='localhost')
```

If the server is on a different machine than the one the model is running on, then replace 'localhost' with the IP address or domain name of that machine.

```
log=ccm.log(trace='192.168.1.10')
log=ccm.log(trace='server.mylab.com')
```

Alternatively, this information can be specified when the model is run. This is done in the following manner

```
python MyModel.py -a localhost
python MyModel.py -a 192.168.1.10
python MyModel.py -a server.mylab.com
```

When the model runs, it will now send all necessary information to the server.

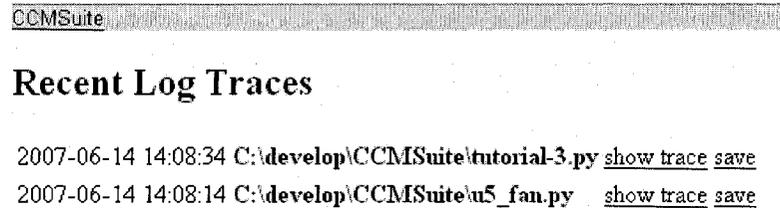
16.4.3 Saving the Model on the Server

The next step is to tell the server to save this model. This is done through the web

browser interface, which can be accessed by opening a web browser and going to.

`http://localhost:8080`

When this happens (assuming the server is running, as per the instructions in the previous section), a display similar to the following should appear.



This display shows all of the models that have been run recently (including the time and date). Clicking on “show trace” will display the same visualization information as seen in section 16.3.

Before the model can be used by the server to do further runs and gather statistical data, the model must be saved. To do this, click on 'save'. The following screen will appear.

Simulation Run Information

Save As:

Save As Summary:

Name C:\develop\CCMSuite\tutorial-3.py

Time 2007-06-14 14:08:34

Parameters

- light=off
- rt=0.161523523847
- score=10
- time=5.835544595

Results

Code

```
import ccm2 as ccm
log=ccm.log()

class ReactionTimeEnvironment(ccm.Model):
    def start(self):
        self.score=0
        self.light='off'
        for i in range(10):
            yield 0.5
            self.light='on'
            timel=self.now()
            yield self.press
            time2=self.now()
            log.rt=time2-timel
            self.light='off'
        self.stop()

    def press(self):
        if self.light=='on':
            self.score=self.score+1
```

This displays all of the information about the model, including its final measurement values. To save it, type a description into the Save As box and press Enter. This description will be the name we will use to refer to the model from this point on.

16.4.4 Running the Model Multiple Times

After the model is saved, you can return to the main page by clicking on “CCMSuite” at the top of the page. This will now look something like the following, where we have saved the model as “Example 3”, since it is based on the example in section 16.2.3.



Recent Log Traces

```
2007-06-14 14:08:34 C:\develop\CCMSuite\tutorial-3.py show trace save  
2007-06-14 14:08:14 C:\develop\CCMSuite\u5_fan.py show trace save
```

Experiments

Example 3 [data](#) [stats](#) [explore](#) 

We now want to run this model many times. Since there is a certain degree of randomness in the reaction times of the model, this will allow us to see the distribution of results.

To do this, we simply click on the “Start Job” button beside Example 3. This tells the server that we would like to gather more data about this model by running it on all of the client computers. When we have enough, we can click on the button again to stop the job.

However, at the moment there are no clients, so no simulations will be run. To set up a client, we go to any computer that has CCMSuite installed (including the one running the server) and run the following command. Note that, just like when telling the server about the model, the last value should be either localhost (if the client is running on the same computer as the server) or the IP address/domain name of the server.

```
python ccm2-jobclient.py -a localhost  
python ccm2-jobclient.py -a 192.168.1.10  
python ccm2-jobclient.py -a server.mylab.com
```

The client will now connect to the server and see if there are any simulations that need to be run. When it finds one, all the data needed to run the model will be transferred to the

client and the model will be run. Once it is finished running, the results will be sent back to the server and stored for later analysis. After this has run for a while and sufficient results have been produced, stop the simulation by clicking on “Stop Job”.

16.5 Statistical Analysis

Once the simulation has been run multiple times, CCMSuite can perform the statistical analysis recommended by the methodology put forth in this thesis. This analysis can be done while the simulations are still being run by the various client computers, or can be done after running the simulations for a sufficient number of runs.

16.5.1 Viewing the Raw Data

By clicking on the “data” link beside the name of the model, one can view the raw data from the simulations. This shows a simple table with each row representing one run of the simulation, and each column showing the various measured values in the simulation. There is also a “csv” link which provides the same data but in a standard “Comma-Separated Values” format, which can be used with any standard statistical analysis program (such as SPSS). Values are shown for everything that appears in the visualizations discussed previously, and include the total amount of time for each run. As always, this is the amount of simulated time built up from the time requirements of the individual steps within the model, not the actually amount of real-world time it takes to run the simulation.

CCMSuite

Experiment Example 3

csv

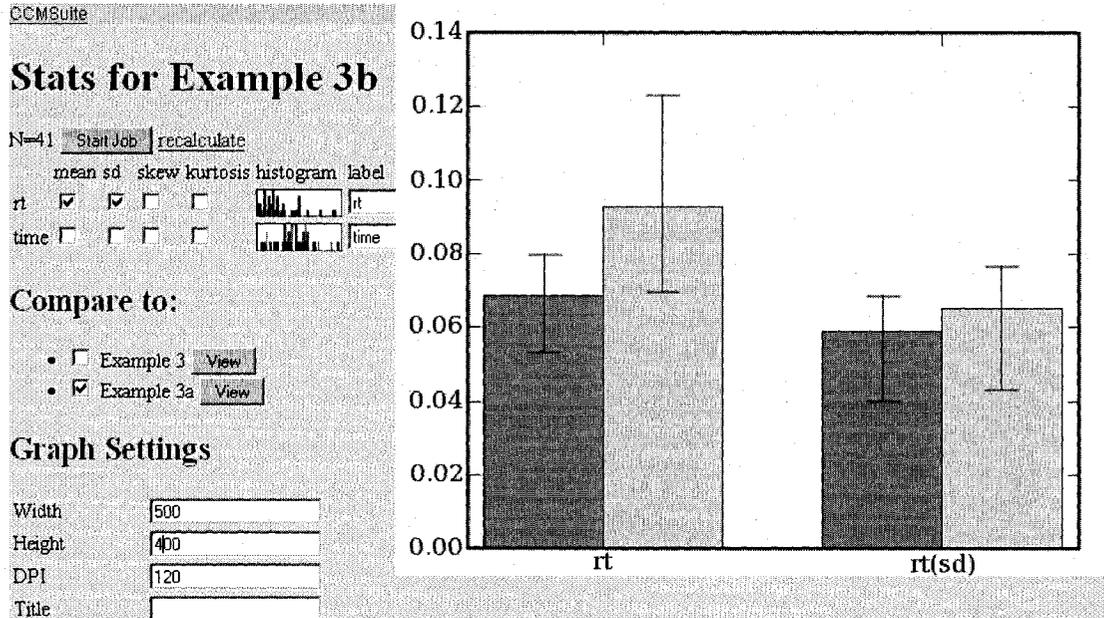
light	rt	score	time
off	0.161523523847	10	5.835544595
off	0.066624577615	10	5.88360540586
off	0.0136352266217	10	5.59431328184
off	0.120196849337	10	5.66564659397
off	0.102023475342	10	5.71992232627
off	0.0290993749195	10	5.74190903116
off	0.0879975850231	10	5.78562539795
off	0.13355168575	10	5.94469226406
off	0.155586886191	10	5.5529476425
off	0.041272643781	10	5.50785826453
off	0.0522178961515	10	5.49000887848
off	0.064245315891	10	5.81098112248
off	0.115272132965	10	5.76876646871
off	0.010111510827	10	5.3538068385

16.5.2 Installing Matplotlib

Since the statistical analysis system needs to make visual graphs of the resulting data, it is also necessary to install the Python module Matplotlib, which is freely available at <http://matplotlib.sourceforge.net/> and also requires the Python Imaging Library, freely available at <http://www.pythonware.com/products/pil/>.

16.5.3 Data Analysis

From the main menu, click on the “stats” link beside the simulation you wish to analyze. This will take you to the basic statistics page. Here you can indicate what aspects of the model you wish to analyze and all data and graphs will be generated.



On this screen, the options on the left control which will be shown in the graph on the right. The first row displays the number of simulations represented (N=41 in this case). It also allows you to Start/Stop the job (to run the simulation more times, exactly as in the button on the main screen discussed previously).

Below this is a list of the values that can be graphed for this model. In this case, the only measurements are *rt* (the reaction time) and *time* (which is always the total amount of simulated time for which the model was run). Since these measurements form distributions over the different runs of the simulation (due to randomness within the simulation), there will be many possible aspects of the distribution that could be examined. Currently, CCMSuite only supports the mean, standard deviation (sd), skew, and kurtosis. Choose the values you wish to see graphed and click on those checkboxes. CCMSuite will automatically generate the graphs, including the confidence intervals for each value. There is also a small histogram of the data shown and a box to

allow you to adjust the label for that measurement on the graph.

In the image above, the values are being compared to results from another model (in this case, a model saved as “Example 3a”). This is controlled by clicking the checkboxes in the “Compare to” section.

Below this are the various configuration options for the graph. You can adjust the width and height of the graph (in pixels). You can also adjust the desired DPI (dots per inch) to achieve publication-quality graphs (usually 300 to 600 dpi). Labels can be added to axes, an overall title can be added, and a legend can be shown. The colours can be adjusted, as can the shape of the confidence interval bars, the position of the axes, and the upper and lower bound on the y-axis. Finally, the method used to calculate the confidence intervals can be changed, allowing for different confidence levels (e.g. 60%, 80%, 99%, 99.99%). The default technique is to use the bootstrap method with 100 samples, but this can be changed to use more or less samples, or to instead make the assumption that the distribution is normal. You can also optionally show the raw data used to generate the graph.

16.6 Parameter Exploration

The previous section demonstrated how to view the overall results of a particular model. It also allowed side-by-side comparisons between models. This approach could be used to examine the effects of parameter adjustment: we would simply create a second copy of a model, change a parameter value, and then run it as before. However, this would be a long and arduous process if there are many different parameter settings to examine.

Instead, there is a separate system in CCMSuite to do a comprehensive analysis of how adjusting parameters of a model affects its behaviour.

16.6.1 Defining Parameters

The first step in exploring parameters is to define exactly what those parameters are.

This is done in the model code itself. Any value in the model can be a parameter. All that needs to happen is that the definition of the parameter needs to be at the very top of the model program.

For example, the model created in section 16.2.3 would pause for somewhere between 0.1 seconds and 0.3 seconds before looking again to see if the light is on. These values can be turned into parameters by defining them at the very top of the program. So, instead of

```
yield self.random.uniform(0.1,0.3)
```

we will define two parameters (`timeToWaitMinimum` and `timeToWaitMaximum`) by adding the following two lines at the beginning of the program (before the “`import ccm2 as ccm`” statement).

```
timeToWaitMinimum=0.1  
timeToWaitMaximum=0.3
```

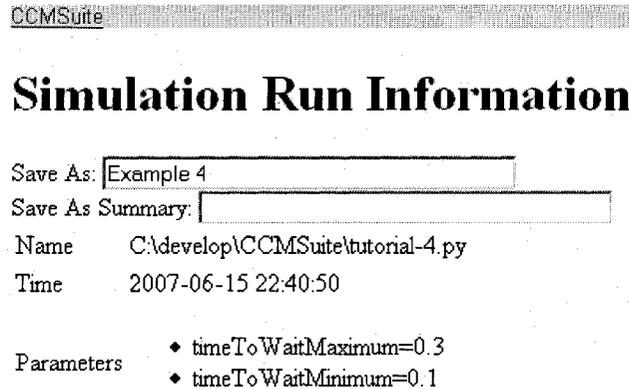
To use these parameters, we also modify the original line as follows

```
yield self.random.uniform(timeToWaitMinimum,timeToWaitMaximum)
```

You can define as many parameters as you like, and parameter values can be numbers or text strings.

16.6.2 Defining an Exploration

Once the parameters are defined in the model, it can be run and saved in the CCMSuite web browser interface exactly as before. When you are saving the model, it should list the parameters it is able to identify.



After it has been saved, return to the main menu and click on “explore” beside the model. This will allow you to create a new exploration.

Here, you must give the exploration a name and then specify what values the parameters should be changed to. For example, the following settings would adjust the timeToWaitMaximum parameter to the values 0.2, 0.3, 0.4, and 0.5.

CCMSuite

Experiment Example 4

Create New Exploration

Name:

Parameter	Value(s)
timeToWaitMaximum	<input type="text" value="0.2.0.3.0.4.0.5"/>
timeToWaitMinimum	<input type="text" value="0.1"/>

You can adjust as many parameters at the same time as you would like. However, it is not recommended that more than four parameters be adjusted at once, since this tends to lead to many thousands of parameter settings. Since each parameter setting must be run multiple times to gather data, having too many different parameter settings makes it more and more difficult to analyze the data.

Create New Exploration

Name:

Parameter	Value(s)
timeToWaitMaximum	<input type="text" value="0.3.0.4.0.5.0.6.0.7"/>
timeToWaitMinimum	<input type="text" value="0.0.05.0.10.0.15.0.2.0.25"/>

16.6.3 Running an Exploration

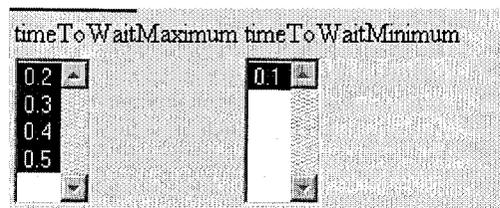
Since an exploration requires examining many different parameter settings, it may take a while to gather all the needed data by running all of the simulations multiple times.

However, the process for doing so is the same as for analyzing a single model. First, go to the “explore” page and click on the exploration (created in the last section) that you wish to examine.

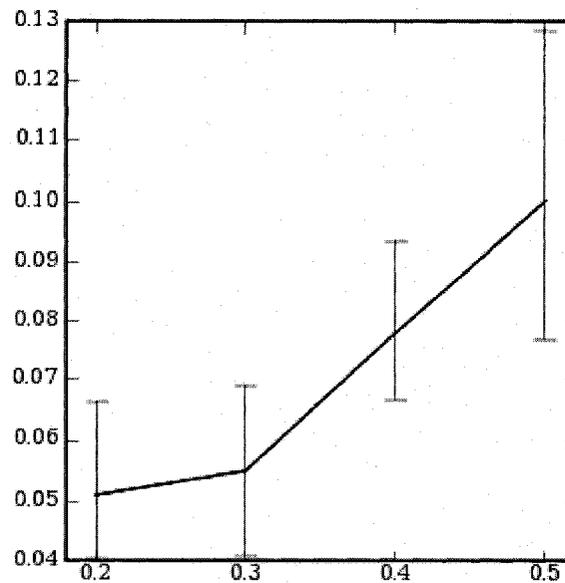
This will take you to the exploration view page. However, since no results have yet been gathered, no information will be available here. To start running the simulations, click on “Start Job”. This will use the same client/server system as discussed previously (section 16.4.4), but will vary the parameter settings as desired.

16.6.4 Analyzing Single Parameters

Once the simulation job has run for a while (or even while it is still running), the analysis of the data can begin. The first step is to specify what values for what parameters should be displayed. This is done by selecting the values under each parameter label. You can use the Ctrl key to select multiple values individually, and the Shift key to select whole ranges of values. By default, all the values of the first parameter that changes will be selected.

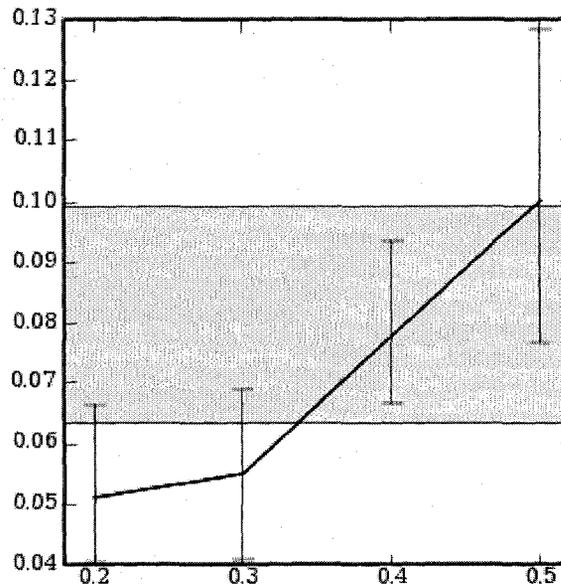


Next, we must select what measurements to display for those parameters. This is done in the measurements section. Multiple measures can be chosen. Once these have been chosen, the graph on the right side of the screen should appear. In this case, we have chosen to graph the rt (reaction time) measurement. If multiple measurements are chosen, they will be drawn as separate lines.

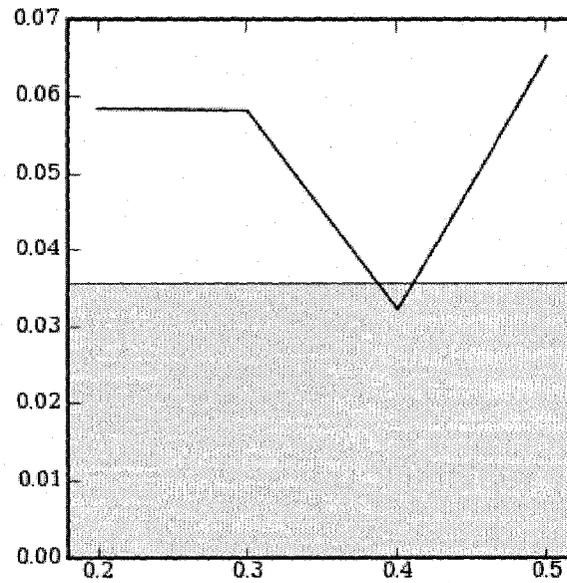


The various controls at the bottom of the page affect the display of the graph. This includes the width, height, dots per inch (dpi), labels, line colours, line thicknesses, confidence intervals, and so on.

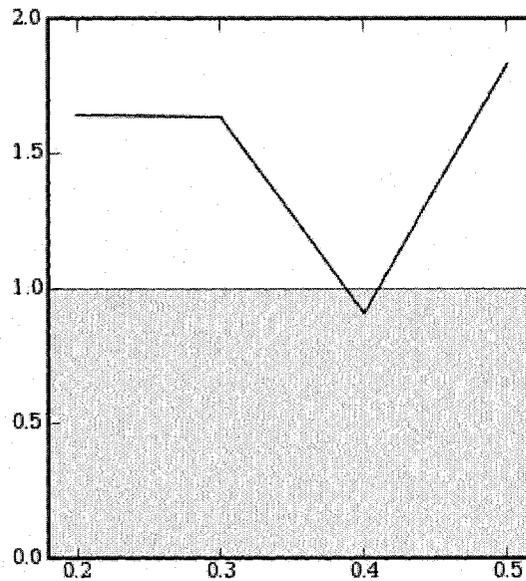
Often, we want to compare the results from the model with those of another model (or the real-world experimental results). This is done with the “Comparison” controls. The first one selects which other data set to compare to. Here, we compare to the Example 3 data, which is shown as a grey bar across the graph. This is the 95% confidence interval from that other set of data. In other words, the comparison value can be said to be within that range (with 95% confidence).



We can also perform Equivalence Testing (see chapter 4) with the comparison. By shifting the “Comparison as” setting to “Maximum Likely Difference”, CCMSuite will perform the basic version of the Equivalence test on each point. This will determine how wrong each of the parameter settings could be. As discussed in chapter 4, this equivalence measure may be reduced by running more simulations (thus decreasing the size of the confidence intervals). However, once a model is known to have less maximum error than the size of the comparison confidence interval, then the model is as good as it can be known to be. In other words, any parameter setting which is drawn inside the grey area in this graph is known to not be statistically significantly different from the comparison data. This identifies the range of parameters which give good models. In the following graph, only the model at 0.4 is known to be a possible match.

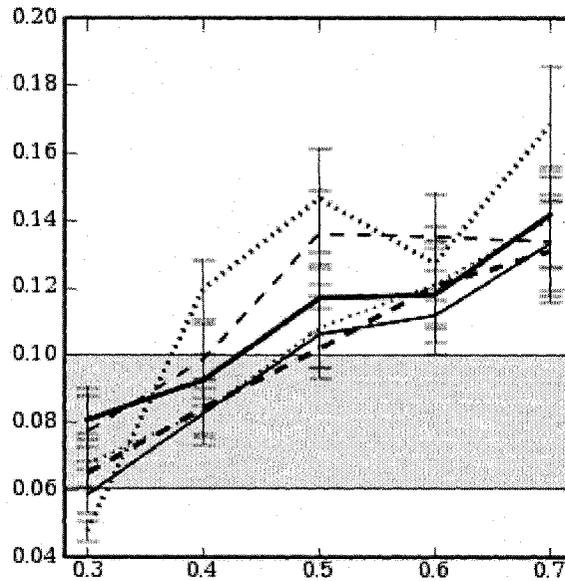


For more standardized comparisons, we can also switch to the Relativized Equivalence measure, by changing “Comparison as” to “Relative Maximum Likely Difference”. This simply rescales the above graph so that the critical value for the model is always 1.0. In this mode, if multiple measurements are chosen, they will all be rescaled before taking the maximum (as discussed in chapter 4).

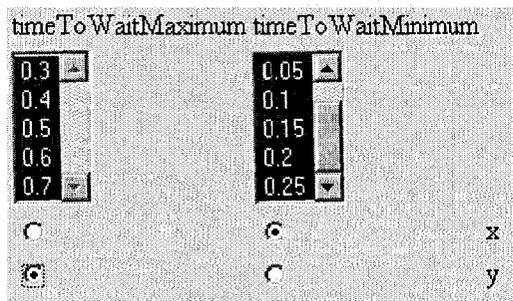


16.6.5 Analyzing Multiple Parameters

In the previous section, we allowed one parameter to vary while keeping all the others at the same value. CCMSuite also allows us to examine what happens when two parameters vary at once. The first step in doing this is to select what parameters to analyze, exactly as we did in the previous section. By selecting multiple values for more than one parameter, we get the following sort of graph.

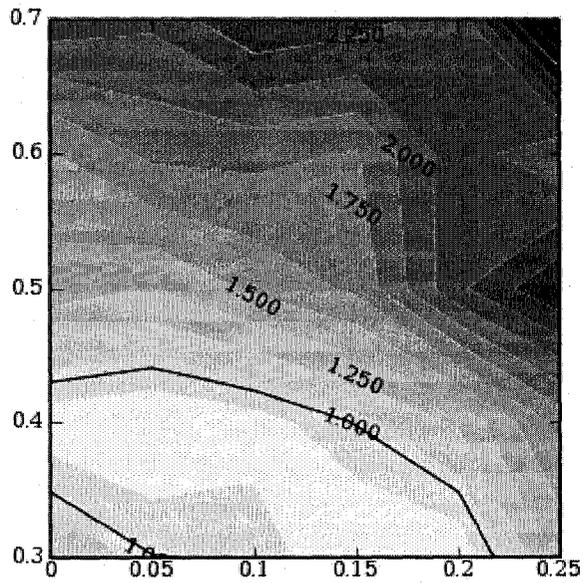


This is not a particularly readable way of displaying information on these different models. So, instead of creating the line graphs we have seen so far, we can create contour plots. In these contour plots we allow one parameter to change along the x-axis, and a different parameter to change along the y-axis. This is chosen by indicating which parameter should be the x-axis and which parameter should be the y-axis.



This leads to the following contour plot. Here, the parameter *timeToWaitMinimum* varies along the x-axis, and *timeToWaitMaximum* varies along the y-axis. The thick lines indicate the space of parameters all of which have a relativized equivalence less than 1.0. In other words, all of the models within that region are equally good models of the

phenomenon.



Many parameters can be adjusted for the contour plots, including the colour (here they have been set to “gray_r”), the number of contour levels, and the overall range.

17 Appendix B: The Models

This appendix provides the source code for all of the models used in this thesis. They are all written in the Python programming language, and make use of CCMSuite. All of these models are also available on-line at <http://ccmlab.ca/ccmsuite.html>. These models are freely available for use by others, and are released under the GNU General Public License.

In keeping with the requirements for CCMSuite, each model begins with the definition of the parameters for the model. These are all adjustable via the CCMSuite parameter exploration system. These models can also be run on their own, and the parameters can be adjusted by directly modifying the settings in the source code.

17.1 Coin Flipping

This is the simple demonstration model used in section 4.3.2. It simply generates a 1 or a 0 with a probability that is controlled by the parameter `prob`. It is such a small model that it does not have a separate environment.

```
# parameters
prob=0.5                                # chance of getting heads

import random                            # gives access to random numbers
import ccm2 as ccm                       # gives access to CCMSuite
log=ccm.log()                            # creates the log to record measurements

if random.random()<prob:
    log.heads=1                          # record a head
else:
    log.heads=0                          # record a tail
```

17.2 Basic Repeated Binary Choice

In chapter 7, the RELACS model with a fixed set of parameters and a single measurement was used to demonstrate how the modelling methodology given in this

thesis would be applied in the most simplified of cases. This demonstrates the creation of a model and an environment, and is the basis of much of the other modelling seen here.

```
import ccm2 as ccm
log=ccm.log()

class Environment(ccm.Model):
    """
    An implementation of the environment for Experiment 2 in (Erev & Barron, 2005)

    This environment model waits for either pressA or pressB to occur. If
    A is pressed, then a reward of 11 is given. If B is pressed, then there
    is a 50% chance of a reward of 1 and a 50% chance of a reward of 19. The
    measurement is the number of times A is pressed in the second block of
    200 trials.
    """

    def start(self):
        # do 2 blocks of 100 trials
        for block in [1,2]:
            self.countA=0
            for i in range(100):
                yield self.pressA,self.pressB # wait until A or B is pressed
            if block==2:
                # if we're in the 2nd block, record the percentage of 'A's
                log.Pmax2=self.countA/100.0
            self.stop()

    def pressA(self):
        self.countA+=1 # count the number of 'A's
        self.reward=11

    def pressB(self):
        if self.random.random()<0.5: # 50% chance
            self.reward=1
        else:
            self.reward=19

import ccm2.lib.relacs
class RELACSModel(ccm.Model):
    """
    A model which uses the RELACS algorithm to choose which of 2 buttons to press

    This defines how the core general-purpose RELACS model (which could be used
    for any reinforcement learning task) can be hooked up to the button pressing
    environment. The actual implementation of the RELACS algorithm is part of the
    CCMSuite, but not shown here.
    """

    def start(self):
        """
        Create a RELACS model with the basic parameters.

        Note that the model requires the expected payoff (the average reward) and
        the expected payoff difference (the average difference between the reward
        and the average reward). This is not information known to the participant
        at the beginning of the experiment, making this a non-process model. That
        is, the steps inside the model are not constrained to possibly correspond
        to the steps in the real system.
        """
        relacs=ccm2.lib.relacs.RELACS(
            expectedPayoff=10.5,expectedPayoffDifference=4.75,
            lambda=8,alpha=0.00125,beta=0.2,kappa=4)
```

```

# repeat until the simulation stops
while True:
    # wait for a second between button presses
    yield 1
    # use the RELACS algorithm to choose an option
    choice=relacs.choose('AB')

    # press the chosen button
    if choice=='A': self.parent.pressA()
    if choice=='B': self.parent.pressB()

    # after pressing the button, learn from the given reward
    relacs.feedback([(choice,self.parent.reward)],choice)

e=Environment()           # make the environment
e.m=RELACSModel()        # put the model in the environment
e.run()                   # run the model

```

17.3 Repeated Binary Choice with Parameters

In chapter 8, model parameters were introduced. The four parameters in RELACS were defined so that the CCMSuite system could explore the effects of changing the parameters. The only difference between this model and the previous one is this parameter definition.

```

# the four RELACS parameters
Lambda=8
Alpha=0.00125
Beta=0.2
Kappa=4

import ccm2 as ccm
log=ccm.log()

class Environment(ccm.Model):
    """
    An implementation of the environment for Experiment 2 in (Erev & Barron, 2005)

    This environment model waits for either pressA or pressB to occur. If
    A is pressed, then a reward of 11 is given. If B is pressed, then there
    is a 50% chance of a reward of 1 and a 50% chance of a reward of 19. The
    measurement is the number of times A is pressed in the second block of
    200 trials.
    """

    def start(self):
        # do 2 blocks of 100 trials
        for block in [1,2]:
            self.countA=0
            for i in range(100):
                yield self.pressA,self.pressB # wait until A or B is pressed
            if block==2:
                # if we're in the 2nd block, record the percentage of 'A's
                log.Pmax2=self.countA/100.0
            self.stop()

    def pressA(self):
        self.countA+=1 # count the number of 'A's

```

```

        self.reward=11

    def pressB(self):
        if self.random.random()<0.5: # 50% chance
            self.reward=1
        else:
            self.reward=19

import ccm2.lib.relacs
class RELACSModel(ccm.Model):
    """
    A model which uses the RELACS algorithm to choose which of 2 buttons to press

    This defines how the core general-purpose RELACS model (which could be used
    for any reinforcement learning task) can be hooked up to the button pressing
    environment. The actual implementation of the RELACS algorithm is part of the
    CCMSuite, but not shown here.
    """
    def start(self):
        """
        Create a RELACS model with the basic parameters.

        Note that the model requires the expected payoff (the average reward) and
        the expected payoff difference (the average difference between the reward
        and the average reward). This is not information known to the participant
        at the beginning of the experiment, making this a non-process model. That
        is, the steps inside the model are not constrained to possibly correspond
        to the steps in the real system.
        """
        relacs=ccm2.lib.relacs.RELACS(
            expectedPayoff=10.5,expectedPayoffDifference=4.75,
            lambd=Lambda,alpha=Alpha,beta=Beta,kappa=Kappa)
            # this uses the parameters defined at the top of the file

        # repeat until the simulation stops
        while True:
            # wait for a second between button presses
            yield 1
            # use the RELACS algorithm to choose an option
            choice=relacs.choose('AB')

            # press the chosen button
            if choice=='A': self.parent.pressA()
            if choice=='B': self.parent.pressB()

            # after pressing the button, learn from the given reward
            relacs.feedback([(choice,self.parent.reward)],choice)

e=Environment() # make the environment
e.m=RELACSModel() # put the model in the environment
e.run() # run the model

```

17.4 Repeated Binary Choice with Multiple Measures

In section 9.1, the environment was expanded to include first 3 new measures and then the rest of the core experiments from (Erev & Barron, 2005). To do this, the environment was modified to be customizable for each situation.

```
# define the parameters
```

```

L=8
a=0.00125
B=0.2
K=4

import ccm2 as ccm
log=ccm.log()
import random

class ErevEnv(ccm.Model):
    """
    A basic environment for all of the Erev & Barron experiments

    This forms the basis for all 40 experiments listed in the
    (Erev & Barron, 2005)
    """
    ready=False # signals whether it is time for the participant to respond
    weight=1 # monetary value for a single reward point
    rewardA=None # reward for pressing A (for complete information paradigm)
    rewardB=None # reward for pressing B (for complete information paradigm)

    def start(self):
        self.countAs() # start a separate system to count A presses
        for block in [1,2]: # only doing 2 blocks, since data is only
            # available for 2nd block
                self.countA=0
                for i in range(100):
                    self.ready=True # signal that participant
                    # should respond now
                    yield self.pressA,self.pressB # wait till a button is pressed
                    self.ready=False
                    yield 1.0 # wait 1 second before the
                    # next response
                # if we're at the second block, record the percentage of 'A's
                if block==2:
                    log.erev[self.id]=self.countA/100.0
        self.stop()

    def countAs(self):
        while True: # repeat while the simulation runs
            yield self.pressA # wait for pressing 'A'
            if self.ready: self.countA+=1 # increase count (if they
            # should be responding now)

class ErevCompleteEnv(ErevEnv):
    """
    The Complete Information paradigm.

    The extends the ErevEnv model so that the information about what
    _would_ have been the reward if the other button is pressed also
    is available.
    """
    def pressA(self):
        self.rewards() # calculate both rewards
        self.reward=self.rewardA # but you actually get reward A
    def pressB(self):
        self.rewards() # calculate both rewards
        self.reward=self.rewardB # but you actually get reward B

# To customize the above environment for the 40 different experimental
# situations, we need the following helper functions for defining the
# probability distributions of the rewards.

# return a with probability p, otherwise b
def bias(a,b,p):
    if random.random()<p: return a

```

```

    else: return b

# a truncated normal distribution
def TN(m, sd):
    x=random.gauss(m, sd)
    if x<0: x=0.0
    return x

# a normal distribution
N=random.gauss

# return a,b with probability p, otherwise b,a
def prob(a,b,p):
    if random.random()<p: return (a,b)
    else: return (b,a)

# do the same as prob() but return 0 with
# probability p2
def prob2(a,b,p1,p2):
    if random.random()<p2:
        return prob(a,b,p1)
    else:
        return (0,0)

# Now each of the experimental situations are defined, based on the above
class Env1(ErevEnv):
    id=1
    weight=0.25
    avgReward=10.5
    avgDiff=0.5
    def pressA(self): self.reward=11
    def pressB(self): self.reward=10

class Env2(ErevEnv):
    id=2
    weight=0.25
    avgReward=10.5
    avgDiff=4.75
    def pressA(self): self.reward=11
    def pressB(self): self.reward=random.choice([19,1])

class Env3(ErevEnv):
    id=3
    weight=0.25
    avgReward=10.5
    avgDiff=5.25
    def pressA(self): self.reward=random.choice([21,1])
    def pressB(self): self.reward=10

class Env4(ErevEnv):
    id=4
    weight=0.25
    avgReward=-10.5
    avgDiff=0.5
    def pressA(self): self.reward=-10
    def pressB(self): self.reward=-11

class Env5(ErevEnv):
    id=5
    weight=0.25
    avgReward=-10.5
    avgDiff=5.25
    def pressA(self): self.reward=-10
    def pressB(self): self.reward=random.choice([-21,-1])

class Env6(ErevEnv):

```

```

    id=6
    weight=0.25
    avgReward=-10.5
    avgDiff=4.75
    def pressA(self): self.reward=random.choice([-19,-1])
    def pressB(self): self.reward=-11

class Env7(ErevCompleteEnv):
    id=7
    weight=0.25
    avgReward=10.5
    avgDiff=0.5
    def rewards(self):
        self.rewardA=11
        self.rewardB=10

class Env8(ErevCompleteEnv):
    id=8
    weight=0.25
    avgReward=10.5
    avgDiff=5.25
    def rewards(self):
        self.rewardA=random.choice([21,1])
        self.rewardB=10

class Env9(ErevCompleteEnv):
    id=9
    weight=0.25
    avgReward=-10.5
    avgDiff=0.5
    def rewards(self):
        self.rewardA=-10
        self.rewardB=-11

class Env10(ErevCompleteEnv):
    id=10
    weight=0.25
    avgReward=-10.5
    avgDiff=5.25
    def rewards(self):
        self.rewardA=-10
        self.rewardB=random.choice([-21,-1])

class Env11(ErevEnv):
    id=11
    weight=0.15
    avgReward=19.5
    avgDiff=2.68922
    def pressA(self): self.reward=random.gauss(21,3)
    def pressB(self): self.reward=random.gauss(18,3)

class Env12(ErevEnv):
    id=12
    weight=0.15
    avgReward=19.5
    avgDiff=3.94526
    def pressA(self): self.reward=random.gauss(21,3)
    def pressB(self): self.reward=random.gauss(18,3)+random.choice([5,-5])

class Env13(ErevEnv):
    id=13
    weight=0.15
    avgReward=19.5
    avgDiff=3.93985
    def pressA(self): self.reward=random.gauss(21,3)+random.choice([5,-5])
    def pressB(self): self.reward=random.gauss(18,3)

class Env14(ErevEnv):

```

```

id=14
weight=0.15
avgReward=19.5
avgDiff=5.20373
def pressA(self): self.reward=random.gauss(21,3)+random.choice([5,-5])
def pressB(self): self.reward=random.gauss(18,3)+random.choice([5,-5])

class Env21(ErevEnv):
    id=21
    weight=0.25
    avgReward=3.1
    avgDiff=0.72
    def pressA(self): self.reward=bias(4,0,0.8)
    def pressB(self): self.reward=3

class Env22(ErevEnv):
    id=22
    weight=0.25
    avgReward=0.775
    avgDiff=1.20125
    def pressA(self): self.reward=bias(4,0,0.2)
    def pressB(self): self.reward=bias(3,0,0.25)

class Env23(ErevEnv):
    id=23
    weight=0.25
    avgReward=3.1
    avgDiff=2.89
    def pressA(self): self.reward=bias(32,0,0.1)
    def pressB(self): self.reward=3

class Env24(ErevEnv):
    id=24
    weight=0.25
    avgReward=0.775
    avgDiff=1.336875
    def pressA(self): self.reward=bias(32,0,0.025)
    def pressB(self): self.reward=bias(3,0,0.25)

class Env25(ErevEnv):
    id=25
    weight=0.25
    avgReward=-3.1
    avgDiff=2.89
    def pressA(self): self.reward=-3
    def pressB(self): self.reward=bias(-32,0,0.1)

class Env26(ErevEnv):
    id=26
    weight=0.25
    avgReward=1262.5
    avgDiff=160.98664
    def pressA(self): self.reward=N(100,354)
    def pressB(self): self.reward=TN(25,17.7)

class Env27(ErevEnv):
    id=27
    weight=0.25
    avgReward=1262.5
    avgDiff=160.98664
    def pressA(self): self.reward=N(1300,354)
    def pressB(self): self.reward=N(1225,17.7)

class Env28(ErevEnv):
    id=28
    weight=0.25
    avgReward=1262.5
    avgDiff=37.70958

```

```

    def pressA(self): self.reward=N(1300,17.7)
    def pressB(self): self.reward=N(1225,17.7)

class Env31(ErevCompleteEnv):
    id=31
    weight=0.25
    avgReward=4
    avgDiff=2
    def rewards(self):
        self.rewardA,self.rewardB=prob( 6, 2, 0.7)

class Env32(ErevCompleteEnv):
    id=32
    weight=0.25
    avgReward=2
    avgDiff=2
    def rewards(self):
        self.rewardA,self.rewardB=prob( 4, 0, 0.7)

class Env33(ErevCompleteEnv):
    id=33
    weight=0.25
    avgReward=0
    avgDiff=2
    def rewards(self):
        self.rewardA,self.rewardB=prob( 2, -2, 0.7)

class Env34(ErevCompleteEnv):
    id=34
    weight=0.25
    avgReward=-2
    avgDiff=2
    def rewards(self):
        self.rewardA,self.rewardB=prob( 0, -4, 0.7)

class Env35(ErevCompleteEnv):
    id=35
    weight=0.25
    avgReward=-4
    avgDiff=2
    def rewards(self):
        self.rewardA,self.rewardB=prob( -2, -6, 0.7)

class Env36(ErevCompleteEnv):
    id=36
    weight=0.25
    avgReward=3.6
    avgDiff=2.16
    def rewards(self):
        self.rewardA,self.rewardB=prob2(6,2,0.7,0.9)

class Env37(ErevCompleteEnv):
    id=37
    weight=0.25
    avgReward=-3.6
    avgDiff=2.16
    def rewards(self):
        self.rewardA,self.rewardB=prob2(-2,-6,0.7,0.9)

class Env38(ErevEnv):
    id=38
    weight=0.25
    avgReward=3.6
    avgDiff=2.16
    def pressA(self): self.reward=prob2(6,2,0.7,0.9)[0]
    def pressB(self): self.reward=prob2(6,2,0.7,0.9)[1]

class Env39(ErevEnv):

```

```

id=39
weight=0.25
avgReward=-3.6
avgDiff=2.16
def pressA(self): self.reward=prob2(-2,-6,0.7,0.9)[0]
def pressB(self): self.reward=prob2(-2,-6,0.7,0.9)[1]

class Env40 (ErevEnv):
    id=40
    weight=0.25
    avgReward=-3.1
    avgDiff=0.72
    def pressA(self): self.reward=-3
    def pressB(self): self.reward=bias(-4,0,0.8)

import ccm2.lib.relacs
class RELACSModel(ccm.ProductionSystem):

    # initialize the RELACS algorithm
    def start(self):
        yield 0.01
        self.relacs=ccm2.lib.relacs.RELACS(
            expectedPayoff=self.parent.avgReward,
            expectedPayoffDifference=self.parent.avgDiff,
            lambda=L,alpha=a,beta=B,kappa=K)

    # whenever the environment is ready, do this
    def press(top='ready:True'):
        choice=self.relacs.choose('AB')
        if choice=='A': self.parent.pressA()
        if choice=='B': self.parent.pressB()
        if self.parent.rewardA is not None:
            self.relacs.feedback([('A',self.parent.rewardA),
                                ('B',self.parent.rewardB)],choice)
        else:
            self.relacs.feedback([(choice,self.parent.reward)],choice)

# now create each environment in turn
all=[Env1,Env2,Env3,Env4,Env5,Env6,Env7,Env8,Env9,Env10,Env11,Env12,Env13,
     Env14,Env21,Env22,Env23,Env24,Env25,Env26,Env27,Env28,Env31,Env32,Env33,
     Env34,Env35,Env36,Env37,Env38,Env39,Env40]
for EnvClass in all:
    e=EnvClass() # create the environment
    e.m=RELACSModel() # put a new version of the model in the environment
    e.run() # run it

```

17.5 Peer Groups

This model is of children interacting in groups, and gradually changing how much they like and dislike each other. Based on this information, individuals are classified as Popular, Rejected, Controversial, Neglected, or Average using the CDC sociometric measurement method.

```

groupCount=100 # number of classrooms
groupSize=30 # number of students per classroom

```

```

time1=100          # number of interactions before first CDC measurement
time2=10          # number of interactions before second CDC measurement

import ccm2 as ccm
import ccm.pyutils.stats as stats
import random

log=ccm.log()

class Agent:
    """
    An individual in a classroom.

    This is a random model, where individuals randomly choose who they
    like and who they dislike.
    """
    def doPreferences(self,others):
        likes=random.sample(others,3)
        dislikes=random.sample(others,3)
        return likes,dislikes

class Group:
    """
    A classroom.

    A collection of individuals who can interact with each other.
    """
    def __init__(self):
        # create all the individuals
        self.agent=[Agent() for x in range(groupSize)]

        # do the measurement and calculate acceptance,
        # rejection, preference, and impact
    def doPreferences(self):
        for a in self.agent:
            a.acceptance=0
            a.rejection=0
        for a in self.agent:
            others=[x for x in self.agent if x!=a]
            likes,dislikes=a.doPreferences(others)
            for l in likes: l.acceptance+=1
            for d in dislikes: d.rejection+=1
        for a in self.agent:
            a.preference=a.acceptance-a.rejection
            a.impact=a.acceptance+a.rejection

class PeerGroups(ccm.Model):
    def start(self):
        # create the classrooms
        self.groups=[Group() for i in range(groupCount)]

        # wait for a period of time
        yield time1

        # do a CDC measurement
        for g in self.groups:
            g.doPreferences()
            self.doCDC()

        # count numbers in each group
        n=float(groupCount*groupSize)
        log.P=self.counts['P']/n
        log.R=self.counts['R']/n
        log.N=self.counts['N']/n
        log.C=self.counts['C']/n
        log.A=self.counts['A']/n

```

```

# wait again and do another CDC measurement
yield time2
for g in self.groups:
    g.doPreferences()
self.doCDC()

# calculate stability of each group
log.sP=self.stable['P']/float(self.counts['P'])
log.sR=self.stable['R']/float(self.counts['R'])
log.sN=self.stable['N']/float(self.counts['N'])
log.sC=self.stable['C']/float(self.counts['C'])
log.sA=self.stable['A']/float(self.counts['A'])

# follow the CDC algorithm to classify individuals
def doCDC(self):
    # build up lists of preference, rejection, acceptance, and impact
    prf=[]
    rej=[]
    acc=[]
    imp=[]
    for g in self.groups:
        for a in g.agent:
            prf.append(a.preference)
            rej.append(a.rejection)
            acc.append(a.acceptance)
            imp.append(a.impact)

    # get mean and standard deviation of the lists
    pMean=stats.mean(prf)
    pSD=stats.stdev(prf)
    rMean=stats.mean(rej)
    rSD=stats.stdev(rej)
    aMean=stats.mean(acc)
    aSD=stats.stdev(acc)
    iMean=stats.mean(imp)
    iSD=stats.stdev(imp)

    # classify individuals
    self.counts={'A':0,'P':0,'R':0,'N':0,'C':0}
    self.stable={'A':0,'P':0,'R':0,'N':0,'C':0}
    for g in self.groups:
        for a in g.agent:
            if a.preference>=pMean+pSD and a.acceptance>=aMean and a.rejection<=rMean:
                category='P'
            elif a.preference<=pMean-pSD and a.acceptance<=aMean and
                a.rejection>=rMean:
                category='R'
            elif a.impact<=iMean-pSD and a.acceptance==0:
                category='N'
            elif a.impact>=iMean+pSD and a.acceptance>=aMean and a.rejection>=rMean:
                category='C'
            else:
                category='A'
            self.counts[category]+=1
            oldCategory=getattr(a,'category',None)
            if category==oldCategory:
                a.stable=True
                self.stable[category]+=1
            else:
                a.stable=False
            a.category=category

# create and run the model
e=PeerGroups()
e.run()

```

17.6 Repeated Binary Choice using Cognitive Architectures

Two types of models based on an existing cognitive architecture (ACT-R) were created for the repeated binary choice task. Since the environment for this task is identical to the one shown above in section 17.4, it will not be repeated here. However, the ACT-R based models are provided.

The first model uses the production utility learning systems in ACT-R to choose between pressing A or B. There are four different systems used, and varying amounts of random noise.

```
noise=0.03                # amount of random variation
pmrule='PGC'              # which rule to use

from ccm2.lib.actr import *
# list of the four different procedural memory rules in ACT-R
PMRules={
  'PGC':PMPGC,            # the standard PG-C rule
  'PGCSW':PMPGCSuccessWeighted, # variant from (Gray, Schoelles, & Sims, 2005)
  'PGCMW':PMPGCMixedWeighted,  # variant from (Gray, Schoelles, & Sims, 2005)
  'New':PMNew}            # new utility learning system

class ACTRPMModel (ACTR):
  pmnoise=PMNoise(noise=noise) # for random variation
  pmpgc=PMRules[pmrule]()      # the chosen utility learning system

  # The two possible actions. The production utility learning system will
  # choose from these based on what it has learned about previous rewards.
  def pressA(top='ready:True'):
    self.parent.pressA()
    self.reward(self.parent.reward)
  def pressB(top='ready:True'):
    self.parent.pressB()
    self.reward(self.parent.reward)
```

The second approach to using the ACT-R cognitive architecture was to make use of the declarative memory system to learn sequential dependencies. This involves remembering particular patterns of events and using them to predict what will happen next.

```
noise=0.3                # amount of random noise
lag=3                    # amount of context (previous actions) to use
memSize=10               # size of memory for quality of reward
threshold=0              # distance from median reward needed to be good or bad

from ccm2.lib.actr import *
```

```

class Numerical(ccm.Model):
    """
    A numerical estimation module.

    This provides the ability to say whether a given reward
    is 'good' or 'bad' based on the previous rewards seen.
    """
    def __init__(self, size=20, threshold=0):
        ccm.Model.__init__(self)
        self.history=[]
        self.size=size           # number of items to remember
        self.threshold=threshold # how far from the median is needed for good/bad
        self.good=False
        self.bad=False

    # determine if a value is good or bad, in previous context
    def evaluate(self, value):
        if len(self.history)==0:
            self.good=True
            self.bad=True
        else:
            h=list(self.history)
            h.sort()
            i=int((0.5+self.threshold)*len(h))
            j=int((0.5-self.threshold)*len(h))
            if value>h[i]: self.good=True
            else: self.good=False
            if value<h[j]: self.bad=True
            else: self.bad=False
        self.history.append(value)
        while len(self.history)>self.size:
            self.history.pop(0)

# the actual ACT-R model
class ACTRDMMModelLag1(ACTR):
    # define the ACT-R modules to use
    retrieval=Buffer()
    dm=Memory(retrieval)
    dmnoise=DMNoise(dm, noise=noise)
    dmb1=DMBaseLevel(dm)
    goal=Buffer()
    num=Numerical(size=memSize, threshold=threshold)

    def init():
        goal.set('mode:choose al:X action:None reward:None')

    # when we need to choose, try to remember something with context
    # that is similar to now
    def choose(goal='mode:choose al:?al', top='ready:True'):
        dm.request('history ?quality ?action ?al')
        goal.modify(mode='recalling')

    # if I recall a good event, do the same action as happened then
    def recallActionGood(goal='mode:recalling', retrieval='history good ?action'):
        if action=='A': self.parent.pressA()
        elif action=='B': self.parent.pressB()
        retrieval.clear()
        goal.modify(mode='feedback', action=action)

    # if I recall a bad event, do the opposite action
    def recallActionBad(goal='mode:recalling', retrieval='history bad ?action'):
        if action=='A': self.parent.pressB()
        elif action=='B': self.parent.pressA()
        retrieval.clear()
        goal.modify(mode='feedback', action=action)

    # if I don't recall, do either A or B

```

```

def noRecallActionA(goal='mode:recalling',dm='error:True'):
    self.parent.pressA()
    goal.modify(mode='feedback',action='A')
def noRecallActionB(goal='mode:recalling',dm='error:True'):
    self.parent.pressB()
    goal.modify(mode='feedback',action='B')

# get feedback from the provided reward
def feedback(goal='mode:feedback',top='reward:?reward'):
    num.evaluate(reward)
    goal.modify(mode='remember',reward=reward)

# remember what happened
def rememberGood(goal='mode:remember al:?al action:?action',
                 num='good:True'):
    dm.add('history good ?action ?al')
    goal.modify(mode='next')
def rememberBad(goal='mode:remember al:?al action:?action',
                num='bad:True'):
    dm.add('history bad ?action ?al')
    goal.modify(mode='next')
def rememberNone(goal='mode:remember action:?action',
                 num='good:False bad:False'):
    goal.modify(mode='next')

# continue to the next case
def next(goal='mode:next action:?action'):
    goal.set('mode:choose al:?action action:None reward:None')

# this modifies the above model to handle a context with lag 2
class ACTRDMMModelLag2(ACTRDMMModelLag1):
    def init():
        goal.set('mode:choose al:X a2:X action:None reward:None')
    def choose(goal='mode:choose al:?al a2:?a2',top='ready:True'):
        dm.request('history ?quality ?action ?al ?a2')
        goal.modify(mode='recalling')
    def rememberGood(goal='mode:remember al:?al a2:?a2 action:?action',
                    num='good:True'):
        dm.add('history good ?action ?al ?a2')
        goal.modify(mode='next')
    def rememberBad(goal='mode:remember al:?al a2:?a2 action:?action',
                   num='bad:True'):
        dm.add('history bad ?action ?al ?a2')
        goal.modify(mode='next')
    def next(goal='mode:next action:?action al:?al'):
        goal.set('mode:choose a2:?al al:?action action:None reward:None')

# this modifies the above model to handle a context with lag 0
class ACTRDMMModelLag0(ACTRDMMModelLag1):
    def init():
        goal.set('mode:choose action:None reward:None')
    def choose(goal='mode:choose',top='ready:True'):
        dm.request('history ?quality ?action')
        goal.modify(mode='recalling')
    def rememberGood(goal='mode:remember action:?action',num='good:True'):
        dm.add('history good ?action')
        goal.modify(mode='next')
    def rememberBad(goal='mode:remember action:?action',num='bad:True'):
        dm.add('history bad ?action')
        goal.modify(mode='next')
    def next(goal='mode:next'):
        goal.set('mode:choose action:None reward:None')

# this modifies the above model to handle a context with lag 3
class ACTRDMMModelLag3(ACTRDMMModelLag1):
    def init():
        goal.set('mode:choose al:X a2:X a3:X action:None reward:None')

```

```
def choose(goal='mode:choose a1:?a1 a2:?a2 a3:?a3',top='ready:True'):
    dm.request('history ?quality ?action ?a1 ?a2 ?a3')
    goal.modify(mode='recalling')
def rememberGood(goal='mode:remember a1:?a1 a2:?a2 a3:?a3 action:?action',
                 num='good:True'):
    dm.add('history good ?action ?a1 ?a2 ?a3')
    goal.modify(mode='next')
def rememberBad(goal='mode:remember a1:?a1 a2:?a2 a3:?a3 action:?action',
                num='bad:True'):
    dm.add('history bad ?action ?a1 ?a2 ?a3')
    goal.modify(mode='next')
def next(goal='mode:next action:?action a1:?a1 a2:?a2'):
    goal.set('mode:choose a3:?a2 a2:?a1 a1:?action action:None reward:None')
```