Evolving Process-Based Models from Psychological Data using Genetic Programming

Peter C. R. Lane[1] and **Peter D. Sozou**[2] and **Mark Addis**[3] and **Fernand Gobet**[4]

Abstract.

The development of computational models to provide explanations of psychological data can be achieved using semi-automated search techniques, such as genetic programming. One challenge with these techniques is to control the type of model that is evolved to be cognitively plausible – a typical problem is that of "bloating", where continued evolution generates models of increasing size without improving overall fitness. In this paper we describe a system for representing psychological data, a class of process-based models, and algorithms for evolving models. We apply this system to the delayed-match-to-sample task. We show how the challenge of bloating may be addressed by extending the fitness function to include measures of cognitive performance.

1 Introduction

Computational modelling traditionally involves the development of cognitive models to suit one or more empirical results. Within psychology, the number of possible results that may be modelled is increasing, leading to a "big data" problem: the large number makes it challenging to develop computational models to understand all of the results [4, 6, 13]. As is occurring in other areas of science, one approach to converting data into theoretical understanding is to use automatic techniques to create suitable models [2, 16]. In this paper, we look at our current progress in constructing a semi-automated system to develop computational models for psychological data, using genetic programming to manage the search for viable models.

We formalise the overall process of model construction into a number of separate stages. The first set of stages provides the raw data for the experiments to be modelled. This includes the definition of the experiment, the measurements to capture, and the primitive operators from which the models will be constructed. This first set of stages requires some work from the researcher for each experiment. The second set of stages is responsible for exploring the space of possible models and selecting the final model or models for further investigation by the researcher. This second stage is built around a genetic-programming system, which uses evolutionary computation to generate and test many thousands of potential models, searching for one which meets the requirements of the experiment and any additional measures of model parsimony or generalisability that the modeller imposes.

One of the standard problems confronting researchers using genetic programming is that of *bloat*, defined as "program growth without (significant) return in terms of fitness." [15, p.101]. As the evolutionary cycles continue, the average size of the individuals within the population tends to increase, without the fitness level generally improving. There are a number of techniques for attempting to tackle bloat. Koza [8] uses a simple cut-off parameter: if any individual is above a given size, it is not added to the evolving population. An alternative, used here, is to modify the fitness function to include a measure of the size of the program.

In this paper, we develop cognitive models of psychological data. Cognitive modelling has a close relationship with artificial intelligence, but also has some additional properties. One of these key differences has been summed up in the following quote:

> AI can have two purposes. One is to use the power of computers to augment human thinking, just as we use motors to augment human or horse power. Robotics and expert systems are major branches of that. The other is to use a computer's artificial intelligence to understand how humans think. In a humanoid way. If you test your programs not merely by what they can accomplish, but how they accomplish it, then you're really doing cognitive science; you're using AI to understand the human mind. (Herbert Simon, in an interview with Doug Stewart. [18])

We can capture 'how' the program is working by comparing fitness against target values obtained from human experiments: an example would be simulated time to run a program compared with human reaction times. These comparisons restrict the space of possible models; for example, optimising the reaction time tends to limit the size and structure of the programs generated by the evolutionary process, because only programs which run close to the target time will produce a good fitness score. These comparisons are in addition to the measured fitness of the model at the specified task. The system then has the character of a multi-objective optimisation system, in which several objective functions must be optimised.

We begin this paper with an explanation of how the experiments and data are defined and presented to the models, before giving an overview of how genetic programming is used to generate viable models. We then explore the problem of "bloating" and the search for parsimonious models in the delayed-match-to-sample task, before finishing with a discussion and conclusion.

[1] School of Computer Science, University of Hertfordshire, College Lane, Hatfield AL10 9AB, United Kingdom, email: peter.lane@bcs.org.uk

[2] Department of Psychological Sciences, University of Liverpool, Bedford Street South, Liverpool L69 7ZA, United Kingdom, email: p.sozou@liverpool.ac.uk

[3] Faculty of Performance, Media and English, Birmingham City University, City North Campus, Perry Barr, Birmingham B42 2SU, United Kingdom, email: mark.addis@bcu.ac.uk

[4] Department of Psychological Sciences, University of Liverpool, Bedford Street South, Liverpool L69 7ZA, United Kingdom, email: fernand.gobet@liverpool.ac.uk

2 Evolving Computational Models

Our proposed system is based on the standard characterisation of scientific research as a heuristic search through a combinatorially large space [10]. The central idea is that computational models are computer programs, which can be tested in a simulated experiment to obtain predicted results. We evaluate the models by comparing their predicted results with the results obtained by humans in a similar experiment. The search space is defined as the space of possible computer programs.

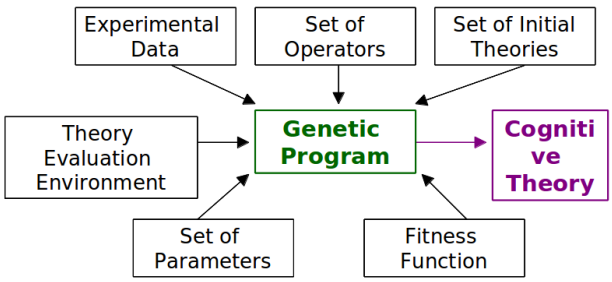


Figure 1. Schematic diagram of system

We divide the system up into a number of stages, as illustrated in Figure 1. The experiment(s) to test are defined in the 'Theory Evaluation Environment,' with the target human data contained in 'Experimental Data.' The space of potential computer programs are defined by the 'Set of Operators.' There are many techniques for managing the search through the space of potential computer programs: in this paper we use a genetic-programming approach [8, 15]. The search process is controlled using the 'Set of Initial Theories' and 'Set of Parameters,' and attempts to optimise the programs against the 'Fitness Function.'

The genetic-programming system works by evolving a population of *candidate models*. An initial population of models is either constructed randomly or seeded from earlier results. Then, processes of mutation and crossover are used to generate a new population, using the fitness function to prefer new models based on the better performing models in the current population. The population is evolved in this way for a number of generations, and the best model in the final population is returned as the 'Cognitive Theory.' We discuss these processes in more detail in the rest of this section.

2.1 Experimental Constraints and Fitness

The Theory Evaluation Environment is responsible for managing the experimental data, and evaluating a fitness function on each of the candidate models. Each set of experimental data is considered as a *constraint* on the model, and is defined from three pieces of information (following [9]):

1. The experimental setting, consisting of its stimuli, and the separation of the stimuli into training and transfer sets.
2. The data collected from the participants.
3. The measure of fit used to compare the model's performance with the participants'.

Constraints can take many forms, and the closeness of the predicted result to the target result can be measured in different ways. In the experiment below, the constraints are the absolute difference from the target average performance and/or reaction time. Constraints may use other error functions or measure qualitative information, such as how close the model's results are to a particular relationship, such as logarithmic.

The genetic-programming system uses a measure of *fitness* to assess the quality of models. In the experiments below, we combine up to three separate components to make up this fitness function. These components are:

1. The overall *performance* of the model at the task;
2. The *reaction time* of the model when doing the task; and
3. The *model complexity*, measured as the size of the program.

The optimisation literature offers a range of techniques for handling separate components in combination (single-objective) or independently (multi-objective); see [9] for an overview. In this paper, we combine components by simply adding the separate components together to make an overall fitness function to optimise.

2.2 Model Definitions

The models are created by combining operators from a Set of Operators; the operators and the rules for their combination form a *theory language* for expressing cognitive models. In principle, these operators can be defined so as to create almost any form of computational model, from symbolic to connectionist. Within this paper, we follow [3] and define a symbolic, process-based class of models. The models have a short-term memory, of three items, and can read from three input positions. The model also has a "current value," used for intermediate results.

The operators are shown in Table 1. The operator `Input1`, for example, reads the current value of input position 1 into the current value. The operator `Compare23` will compare the second and third STM locations, placing true into the current value if they are the same, or false if not. `Prog2` is used to chain two operators together, to perform in sequence.

Each operator has a time cost to it, and the total time cost is used to match the delay in making a decision; the time costs are shown in the second column of Table 1. Notice that some of the operations are repeated for different choices of STM position or input position. For example, `AccessStm1` accesses position 1 of STM, and operators `AccessStm2` and `AccessStm3` are also included to access positions 2 and 3 of STM – these three operations are represented in a single row of the table, to save space. Similarly, there are three comparison operation to compare STM positions 1 and 2, 2 and 3 or 1 and 3.

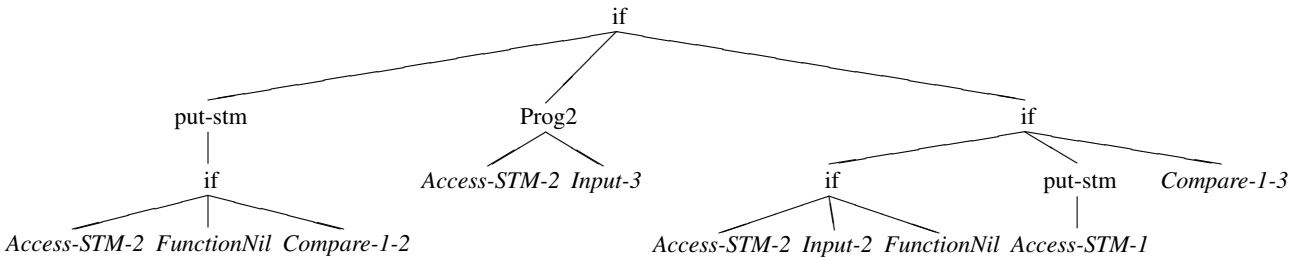
Each model is represented as a program, as shown in Figure 2, which illustrates both the graphical representation for the program, and its Lisp S-expression syntax, as used in the output from the evolutionary system. This program is then "run" in the simulated experiment, to generate a set of simulated results, to be compared with the human data, using the fitness function.

2.3 Search Using Genetic Programming

We manage the search through the space of candidate models using a form of evolutionary computation known as *genetic programming* [8, 15]. Evolutionary computation is an approach to optimisation that applies a Darwinian evolutionary process to the current

Operator	Time (ms)	Description
AccessStm1 (2 or 3)	50	Put item in STM slot 1 (2 or 3) into current value
Compare12 (23 or 13)	200	Current value is true/false if STM item 1/2/3 = item 2/3/1
If	200	Selects between two operators based on current value
Input1 (2 or 3)	200	Read input position 1 (2 or 3) into current value
Nil	50	Set current value to 0 ('false')
Prog2	50	Sequentially do two operators
PutSTM	50	Push current value on to STM

Table 1. Operators used in DMTS models




```
(if (put-stm
      (if Access-STM-2
         FunctionNil
         Compare-1-2))
   (Prog2 Access-STM-2 Input-3)
   (if (if Access-STM-2 Input-2 FunctionNil)
      (put-stm Access-STM-1)
      Compare-1-3))
```

Figure 2. Example of program: The same program is shown in its graphical representation (upper), and as an S-expression.

population of candidate models. Key characteristics of this process are the preferential *selection* of the most fit models for generating the next population. These selected models are then modified using *mutation*, to introduce random changes, and *crossover*, to produce new models from combinations of existing ones. Figure 3 illustrates these two forms of modification:

Mutation takes an existing part of the tree and replaces it with an equivalent element. In the example shown, a '+' operator has replaced the root (top) node of the tree.

Crossover takes two existing trees, selects a sub-tree at random in each, and then creates two new trees by swapping the sub-trees over. In the example shown, the sub-tree `(+ Y 4)` is swapped with the sub-tree `S`.

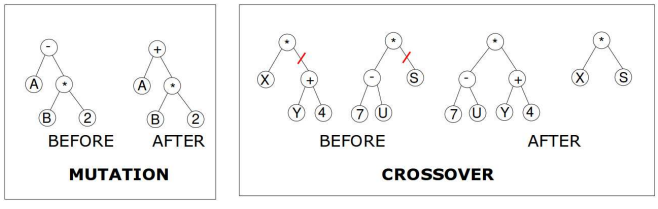


Figure 3. Example of genetic-programming processes of mutation and crossover

The search process begins from an initial population formed from a random sample of models, generated by random combinations of the operators. In each cycle of evolution, the models in the population are evaluated with the fitness function. A new population is then generated using the mutation and crossover processes on selected models from the existing population. Models are preferentially selected from the existing population based on the fitness value. Additionally, the top few models (known as the 'elite') are automatically added to the new population, unchanged. The inherent variability in the process means that two runs of the search process are not guaranteed to produce the same result.

There are a number of parameters which can be adjusted to fine-tune the search for a suitable model. Two of the more important are the number of cycles of evolution performed, and the size of the population at each cycle; in general, larger populations permit greater diversity of models. Other parameters include the number of elite models to retain in the new population, the ways in which mutation and crossover are performed, and the composition of random or seed models used for the first cycle.

We use the ECJ toolkit [12], a Java-based genetic-programming toolkit, in our experiments. This toolkit provides a number of practical advantages, including the ability to run across multiple proces-

sors, and an object-oriented approach which supports a natural subdivision of the system into different modules for programming.

3 Experiment: Delayed Match to Sample

The Delayed Match To Sample (DMTS) task explores processes of short-term memory, categorisation and object recognition. Each trial in the experiment involves presenting a stimulus to the participant. After a delay, two stimuli are presented, one being the first repeated, and the other new. The participant must now identify which was presented first. Numerous studies have used this task, comparing results from different conditions, and looking at which areas of the brain are involved, including [1, 3, 7].

Here, we build on the work of [3] and develop some models of DMTS to optimise not only for performance, but also for reaction time (which was not used in [3]). The aim of the experiment is to explore issues of over-fitting, model parsimony and the problem of bloating.

We use three different measures:

1. *Performance* is measured by calculating the proportion of correct responses out of the total. The absolute difference between this and the target response (from Table 2) forms the performance measure. The smaller this difference is, the 'more fit' the model is.
2. The *size of the models* is measured by counting the number of nodes in the tree, up to a maximum of 1000, and then dividing by 1000. Hence, smaller models produce a smaller value, and hence are 'more fit' than larger models.
3. The *reaction time* is measured by accumulating the simulated time for the operators used when making a decision, and calculating the absolute difference with the target reaction time observed with the human participants (from Table 2).

We run four evolutionary cycles, using four conditions, varying the fitness function in each case:

Condition 1 measures performance only (as in [3]),
Condition 2 measures performance and uses the size of the models as a measure of parsimony,
Condition 3 measures performance and reaction time, and
Condition 4 uses all three measures: performance, reaction time and model size.

The combined fitness function is a real number, formed by adding the contribution from the relevant components in each condition, with 0 being a measure for a 'perfect' model, and larger values representing increasingly unfit models.

3.1 Experimental Setup

We attempt to model the experiment by [1], which used two sets of stimuli: tools and animals. Six stimuli were in each set, and four subjects were presented with sixty trials. Chao et al state there was no significant difference between results for the two sets; as it is difficult in our experiments to make a separation between tools and animals we just use the overall results for tools, as given in Table 2.

The data for the stimuli were simulated with random sequences of the digits 1 to 6. We assume there is no confusion in identifying a given stimulus. A further assumption is that our models are presented with all three stimuli simultaneously, and the task is to determine which of the second and third stimuli is the same as the first. These assumptions (also made in [3]) are reasonable as a test of our methodology, but are different from the original DMTS task and would need to be revisited before making strong claims of theoretical understanding from the models.

	Performance		**Time**	
Stimulus	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*
Tools	95%	$\pm$ 1.2%	767ms	$\pm$ 27.5ms

Table 2. Human performance on DMTS task (percentage correct and reaction time)

The models are defined by the operators given earlier in Table 1.

We used a standard set of parameters for the evolution, with 500 individuals in each population, the best 50 (elite) preserved across generations, and let the system run for 500 generations. Results for each condition were averaged over four runs of each model, to simulate the four participants, and sixty trials for each participant. Finally, results for each condition were averaged over ten different runs.

3.2 Experimental Results

Figure 4 shows a graph of performance against generation for the four conditions; the performance shown is the difference between the performance of the best individual found to that point compared with the target human value. This graph shows how the different achieved performance levels compare. Condition 1 produces the 'best' value for performance, with an almost perfect score; the score is so low, and the models so complex, that condition 1 is a typical case of overfitting. Condition 2 includes the size of the models, and produces a low performance score. Conditions 3 and 4 include the reaction time, which is an additional constraint on the models; this means the models are not so focussed on performance, and hence the scores are not as low as for condition 1. Most of the improvement occurs in the first 100-150 cycles, with only slight improvements found later.

Figure 5 shows a graph of program size against generation for the four conditions; the size shown is the size of the best individual found to that point (averaged over 10 separate runs). This graph shows how the different conditions produce individuals of different sizes. The most noticeable point of this graph is that the size of the models is smallest in conditions 3 and 4, which included a measure of the reaction time. Condition 2, which used a traditional measure of parsimony, also had a dramatic impact on the program size, but not as large as the measure of reaction time. As with performance, most of the changes in program size occur in the first 100 cycles.

Table 3 shows in tabular format the average performance, reaction time and size of the best models created by each of the four conditions, averaged over 10 runs. The performance and reaction-time measures are the absolute differences from the human times, shown in Table 2, and so smaller values are better. This table reinforces the impression from the above two graphs that a good compromise between reduced model size and excellent performance is obtained by introducing the reaction time constraint.

Adding in a measure of parsimony has a much smaller impact. The models generated by conditions 1 and 2 did not evolve against reaction time, but we can see that the use of parsimony has generated models which produce a quicker reaction time. In contrast, those models created by conditions 3 and 4, which included reaction time in the fitness function, produce a tighter fit to reaction time, as may be expected.

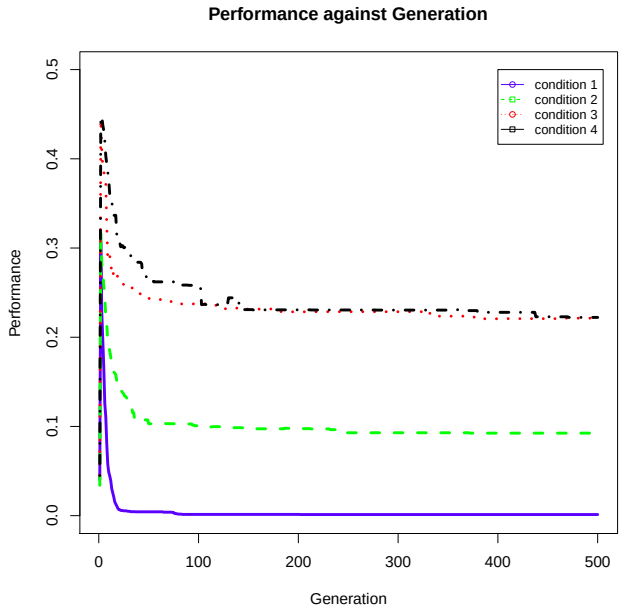


Figure 4. Performance against generation for the four conditions. Results are the mean over 10 runs. (Lower values of performance are better.)

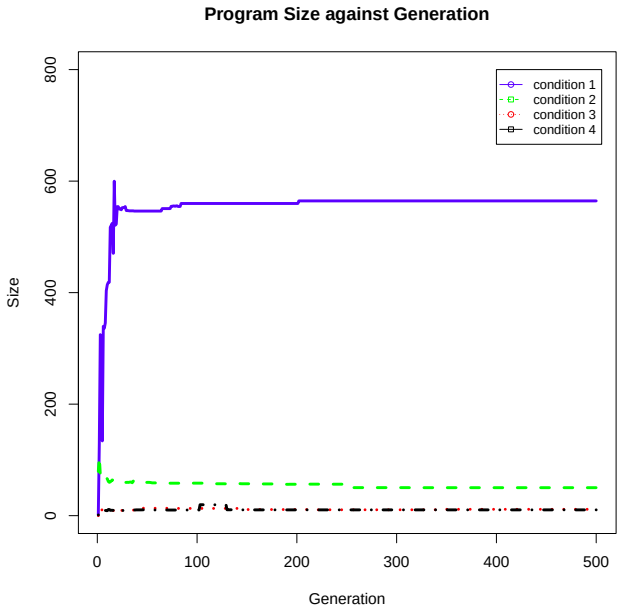


Figure 5. Program size against generation for the four conditions. Results are the mean of the size of the best-performing individual over 10 runs.

	Condition			
Measure	1	2	3	4
Performance	0.001	0.092	0.221	0.222
Reaction Time	4.395	0.672	0.025	0.012
Model Size	564	50.2	11.4	10.4

Table 3. Average results for best individual models created in each condition

3.3 Summary

This experiment is an extension of those previously reported in [3]. The results here differ firstly in that the experiment is implemented in Java using the ECJ toolkit, as described above. Also, the experiments include the timing of decisions as well as overall performance. Finally, larger populations with more cycles were used, and a measure of parsimony, in terms of program size, has been included to address issues of bloating. In comparison with [3] the models generated in condition 1 tend to be larger, which is due to different implementations of mutation and crossover in the genetic-programming libraries used. Also, the results here produce a better overall fitness score.

The main conclusion from this study is that the use of cognitive measures of *how* the model is performing, such as reaction time, can help reduce the overall complexity of the model, whilst retaining an acceptable level of task performance when compared with human performance. In effect, additional measures provide extra constraints on the space of viable models which can be generated from the set of operators. As is evident from the almost identical results of conditions 3 and 4, the inclusion of reaction time makes the need for a direct measure of parsimony, such as program size, unnecessary.

4 Discussion

Many areas of science are developing computational techniques to convert large amounts of raw data into theoretical understanding or practical algorithms. Although this idea has been explored at different times in the history of artificial intelligence [8, 11], ever increasing computational power and new theoretical insights have made significant advances possible in more recent times. Another driver of this recent interest in automatic techniques to analyse data is the increased availability of data in most sciences, due in part to the existence of the internet and in part to the larger amount of published research.

In most cases, the analysis techniques rely on a search over a defined space of potential functions or programs [10]. For example, in robotics, Deisenroth *et al.* [2] use a search over functions defined by Gaussian Processes to efficiently learn algorithms to control complex behaviours. A genetic-programming approach, with some parallels to the one adopted here and in [3], has been used in a physics setting, to find the laws of motion of a pendulum's swing [16].

In this paper, we have examined a genetic-programming approach to developing cognitive theories of psychological behaviour. We have explained the basic principles of genetic programming, the problem domain (DMTS), and the space of potential theories. Our main interest in this paper was to consider the role of different objectives in the fitness function in generating different models, and their performance fitness. The experiment compared models using a simple measure of performance fitness, with models developed additionally with either a constraint on timing or a measure of parsimony or both. Overall, we found that including the constraint on time led to similar benefits to including a measure of parsimony. This result may be interpreted as suggesting that developing cognitive models should use measures based on 'how' the task is performed (such as reaction time), as well as 'what' is done (such as overall task performance).

The optimisation process is better constrained by attempting to develop models which better understand the human mind, using more of the available psychological data.

In future work, we intend to extend the models to tackle problems in areas such as attention, categorisation, and decision making, aiming to find models which will generalise as much as possible across all areas. The framework we have presented here should support such extensions, as its three principal elements have all been separately explored.

First, the genetic-programming system itself is general purpose and robust. Almost two decades of application in many diverse areas has demonstrated its suitability in many different areas [8, 15]. Alternative fitness functions, diversity measures, evolutionary operators and optimisation schemes, such as single or multiple objective, have all been explored in the literature.

Second, there is a prevalence of raw data in each of these areas. As a simple example, the area of categorisation [9, 17] has attracted many variations in the psychological literature, and dozens of different published models.

Third, the general scheme for the models is an adaptation of the general architecture used in symbolic cognitive modelling, as used in systems such as CHREST [5] or Soar [14]. These systems all have some kind of input/output controller, a long-term memory and a short-term memory. Such cognitive architectures already have proven successful in implementing models in areas such as memory recall, perception, problem solving, categorisation and language learning. As we work with more complex domains, we expect to use more of the properties of cognitive architectures within our models, and these prior examples will help guide the choice of operators.

5 Conclusion

The research presented here forms part of a project leading towards the development of programs that automatically and systematically read scientific publications, make summaries of data, and develop theories integrating and explaining the results of a large number of experiments. We believe such programs will be increasingly important to help tackle the "big data" issues in psychology, converting the increasing number of empirical results into theoretical understanding.

6 Acknowledgments

This research was supported by ESRC Grant ES/L003090/1.

REFERENCES

[1] L.L. Chao, J.V. Haxby, and A. Martin, 'Attribute-based neural substrates in temporal cortex for perceiving and knowing about objects', *Nature Neuroscience*, **2**, 913–20, (1999).

[2] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, 'Gaussian processes for data-efficient learning in robotics and control', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2014).

[3] E. Frias-Martinez and F. Gobet, 'Automatic generation of cognitive theories using genetic programming', *Minds and Machines*, **17**, 287–309, (2007).

[4] F. Gobet and P. C. R. Lane, 'A distributed framework for semi-automatically developing architectures of brain and mind', in *Proceedings of the First International Conference on e-Social Science*, (2005).

[5] F. Gobet, P. C. R. Lane, S. J. Croker, P. C-H. Cheng, G. Jones, I. Oliver, and J. M. Pine, 'Chunking mechanisms in human learning', *Trends in Cognitive Sciences*, **5**, 236–243, (2001).

[6] F. Gobet and A. Parker, 'Evolving structure-function mappings in cognitive neuroscience using genetic programming', *Swiss Journal of Psychology*, **64**, 231–239, (2005).

[7] C. Habeck, J. Hilton, E. Zarahn, J. Flynn, J.R. Moeller, and Y. Stern, 'Relation of cognitive reserve and task performance to expression of regional covariance networks in an event-related fMRI study of non-verbal memory', *NeuroImage*, **20**, 1723–33, (2003).

[8] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[9] P. C. R. Lane and F. Gobet, 'Evolving non-dominated parameter sets for computational models from multiple experiments', *Journal of Artificial General Intelligence*, (in press).

[10] P. Langley, H. A. Simon, G. Bradshaw, and J. Zytkow, *Scientific Discovery: Computational Explorations of the Creative Processes*, MIT Press, Cambridge, MA, 1987.

[11] D. B. Lenat, 'Am: an artificial intelligence appraoch to discovery in mathematics as heuristic search', in *Knowledge-Based Systems in Artificial Intelligence*, eds., R. Davis and D. B. Lenat, McGraw-Hill, New York, (1982).

[12] S. Luke. The ECJ owner's manual, 2013.

[13] I. J. Myung and M. A. Pitt, 'Cognitive modeling repository', in *Proceedings of the Thirty-Second Annual Meeting of the Cognitive Science Society*, p. 556, Portland, Oregon, (2010). Lawrence Erlbaum.

[14] A. Newell, *Unified Theories of Cognition*, Harvard University Press, Cambridge, MA, 1990.

[15] R. Poli, W. B. Langdon, and M. F. McPhee, *A Field Guide to Genetic Programming*, Lulu Books, 2008.

[16] M. Schmidt and H. Lipson, 'Distilling free-form natural laws from experimental data', *Science*, **324**, (2009).

[17] J. D. Smith and J. P. Minda, 'Thirty categorization results in search of a model', *Journal of Experimental Psychology: Learning, Memory and Cognition*, **26**, 3–27, (2000).

[18] D. Stewart. Herbert Simon: Thinking machines. Interview conducted June 1994. Transcript available at http://www.astralgia.com/webportfolio/omnimoment/archives/interviews/simon.html, 1997.