Synthesising Cognitive Models with Genetic Programming

Peter Lane[1*], Fernand Gobet[2], Angelo Pirrone[2], Laura Bartlett[2] and Noman Javed[2]
[1]*Department of Computer Science, University of Hertfordshire*
[2]*London School of Economics and Political Science*
*corresponding author: p.c.lane@herts.ac.uk

Developing, understanding and verifying the behaviour of cognitive models is a non-trivial task. A good cognitive model explains and predicts human behaviour in a particular experimental setting. Cognitive models are often in the form of computer programs which need to be designed and written for the given experiment and behaviour: time constraints or natural bias (oversights) often lead to models written by human programmers being constrained to particular theoretical assumptions. We apply program synthesis to this task, introducing novel training and post-processing techniques; our experiments automatically create good quality models, and help visualise the structure of the solution space.

Keywords: cognitive science; computational modelling; decision-making; genetic programming; program synthesis

Introduction

Developing and verifying the behaviour of cognitive models is a non-trivial task. Ideally, a cognitive model will provide some explanation of how a human performs in a particular experimental setting, and even provide predictions for new settings. In many cases cognitive models are based around computer programs which need to be written. The area of program synthesis studies ways to generate executable computer programs from user specifications. In this paper we demonstrate how an evolutionary algorithm can generate programs representing candidate computational models in a typical neuro-scientific experiment. We present techniques to improve the understandability of the resulting programs, which enables their use as the starting point for developing scientific theories.

Proposed System

Our proposed system is based on Genetic Programming [4], a technique which searches a large space of programs for candidate solutions to a given fitness criterion. We have applied our methodology to various tasks, including variants of the Delayed Match To Sample task (DMTS) and a Decision Making task – in this paper, we discuss only the DMTS task. Unique aspects of our system include a phased-evolution system, which aids in finding models with both behaviour and time fitness requirements, and extensive post-processing steps, which reduce the large number of models output by the system to a smaller, more understandable subset, with graphical and text representations. Our approach [2, 5, 6] using GP appears unique in developing cognitive models which focus on symbolic, information-processing [8] explanations of human cognition. This contrasts with many current approaches in artificial intelligence which rely on connectionist (statistical) explanations based on large datasets: a recent study in this area is that of [7].

As an example of program synthesis, our system can be conveniently divided into three parts [3]: the task definition (user intent), to express what makes a good program; a search space of candidate programs; and a search technique (Genetic Programming), to explore the given search space for good programs.

The task studied in this paper is the DMTS task [1], a typical neuroscientific experiment, popular for studies of short-term memory, which tests the accuracy and reaction time for subjects to recognise images. In this experiment a picture is presented for 1 second in the centre of the screen. Then, after a delay of 0.5 seconds, two pictures are presented for 2 seconds, one on the left and the other on the right of the screen. The participant must select which of those two pictures is the same as the first picture.

Although this task is an example of "programming-by-example", where the model must reproduce the example input-output behaviour, the overall quality of the model is not judged on the number of correct input-output pairs. As reported in [1], across the complete set of presentations, human subjects only score 95.7% accuracy, with an average response time of 767ms: the model's accuracy and simulated response times are judged against these values. Such time-dependent models require special attention during the evolution process and, in particular, we have created a novel phased-evolution system which gradually introduces elements of the fitness function over time.

Each individual model is defined by a control program to be interpreted within a simple cognitive architecture: the space of possible control programs is the search space for our system. This architecture has some task-specific input/output components: a set of inputs and a response. It also has some task-independent components: a fixed-size short-term memory (STM), and a working memory. Finally, each model has a clock, to record its current in-task time. The model's control program is composed from a set of operators defining a simple imperative programming language. The model's current working value, STM and clock values can all be manipulated, inputs read and a response prepared: the current response is "made" when the program ends.

Phased Evolution The fitness function is a linear combination of three components: *accuracy*, the proportion of correct responses compared to humans; *response time*, the simulated response time compared to humans; and *program size*, the number of operators in the control program.

Our phased-evolution approach introduces these components gradually. Initially, the GP system is trained using just *accuracy* as the fitness. Once the best model reaches a good level of accuracy (defined as a value less than 0.1), the fitness is changed to be a combination of *accuracy* and *response time*. Thus, the models so far showing accurate results must now additionally make their response in the appropriate amount of time. Again, when the best model reaches a good level of fitness, the fitness is adjusted to be a combination of all three factors, which has the effect of encouraging the evolved models to reduce in size.

Post-Processing In order to reduce the size and complexity of the evolved programs, two post-processing steps are applied. The first simply removes "dead code", defined as code which is not actually called during the running of the program. For example, a condition may always be true and so the else-block of an if-statement may never be used. This can be rewritten as follows:

```
(IF (CONDITION) (SOME-CODE) (UNUSED)) -> (PROG2 (CONDITION) (SOME-CODE))
```

Secondly, some operators may be "masked" by later operators, e.g. the information from looking left may be over-written when subsequently looking right. In this case, the initial look left can have no behavioural effect, except for the simulated time. We can thus rewrite such a case with a special WAIT operator, as follows:

```
(PROG2 (INPUT-LEFT) (INPUT-RIGHT)) -> (PROG2 (WAIT-INPUT) (INPUT-RIGHT))
```

Results and discussion

In one experiment, we used a population size of 500 individuals and 2000 generations, and collected models from six runs.

Fig.1 (a) illustrates the effect of our phased-evolution system. For the first 10 generations, the models are evaluated purely on accuracy: once the best model produces a good accuracy (the red line), the overall fitness drops and the next component of the fitness is introduced – the response time. This second stage lasts up to 60 generations during which period the program size of the best model increases until the response time (the blue line) itself drops, and now the third component can be introduced – program size. The later generations develop models optimised against all three components of fitness in combination, producing small models with a high accuracy and good response time.

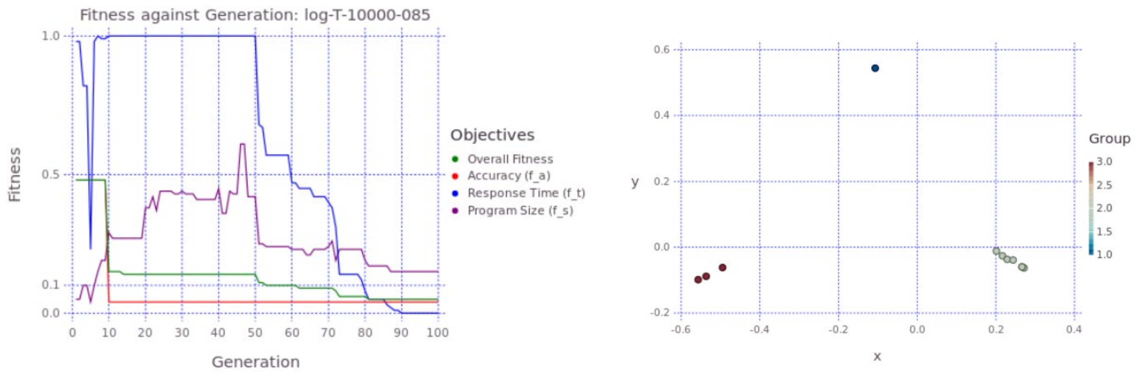

Fig. 1. (a) A graph of fitness against generation, showing how the overall fitness drops in stages, with the separate fitness components being optimized in turn. (b) A scatter plot of similarity of the final models, showing how the models fall into two main groups and one exception.

The GP search technique is used to generate multiple candidate models: each run generates many ‘good’ models (based on a fitness value threshold). A unique aspect of our approach is the amount of post-processing performed on the candidate models, which are otherwise too numerous to analyse and understand. For example (see [6] for details), a typical output of six runs of the GP system produced 1164 distinct models with a good fitness value. By removing bloat, these were reduced to 248 distinct models. We then rewrote semantically equivalent programs to further reduce the number to 11 distinct models. Application of a clustering algorithm (see Fig.1(b)) divides this space of models into two basic groups and one exceptional group – studying individual members of those groups enables a scientist to develop an explanation of behaviour in this domain.

Conclusion

The results obtained so far demonstrate that the approach is successful in synthesizing high-quality cognitive models, based on comparisons with human data. Apart from applying the approach to more complex tasks, further areas for investigation include a co-evolution approach, to optimise the operator time parameters, and domain-specific heuristics for the GP algorithm.

Acknowledgements This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (Grant agreement No. ERC-ADG-835002).

References

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

[2] Frias-Martinez E, Gobet F, Automatic generation of cognitive theories using genetic programming. Minds and Machines 17:287–309, 2007.

[3] Gulwani S, Dimensions in program synthesis. In: Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming, pp 13–24, 2010.

[4] Koza JR, Genetic Programming: On the programming of computers by means of natural selection. MIT Press, Cambridge, MA, 1992.

[5] Lane PCR, Sozou PD, Gobet F, Addis M, Analysing psychological data by evolving computational models. In: Wilhelm A, Kestler H (eds) Analysis of Large and Complex Data. Studies in Classification, Data Analysis, and Knowledge Organization. Springer, Cham, pp 587–97, 2016. https://doi.org/10.1007/978-3-319-25226-1_50

[6] Lane PCR, Bartlett L, Javed N, Pirrone A, Gobet F, Evolving understandable cognitive models. In: Proceedings of the 20th International Conference on Cognitive Modeling, 2022.

[7] Peterson JC, Bourgin DD, Agrawal M, Reichman, D., & Griffiths, T. L., Using large-scale experiments and machine learning to discover theories of human decision-making. Science 372:1209 – 1214, 2021.

[8] Simon HA, Information-processing models of cognition. Annual Review of Psychology 30:363–96, 1981.