Frias-Martinez , E., & Gobet, F. (in press). Automatic generation of cognitive theories using genetic programming. *Minds & Machines*. (The original publication is available at www.springerlink.com.)

Automatic Generation of Cognitive Theories using Genetic Programming

ENRIQUE FRIAS-MARTINEZ
Department of Information Systems & Computing , Brunel University, Uxbridge, UB8 3PH, U.K.

FERNAND GOBET
Centre for Cognition and Neuroimaging, Brunel University, Uxbridge, UB8 3PH, U.K.

Abstract. Cognitive neuroscience is the branch of neuroscience that studies the neural mechanisms underpinning cognition and develops theories explaining them. Within cognitive neuroscience, computational neuroscience focuses on modeling behavior, using theories expressed as computer programs. Up to now, computational theories have been formulated by neuroscientists. In this paper, we present a new approach to theory development in neuroscience: the automatic generation and testing of cognitive theories using genetic programming. Our approach evolves from experimental data cognitive theories that explain "the mental program" that subjects use to solve a specific task. As an example, we have focused on a typical neuroscience experiment, the delayed-match-to-sample (DMTS) task. The main goal of our approach is to develop a tool that neuroscientists can use to develop better cognitive theories.

Key Words: Cognitive Neuroscience, Computational Neuroscience, Automatic Generation of Cognitive Theories, Genetic Programming (GP), Delayed-Match-To-Sample (DMTS).

1 Introduction

Cognitive neuroscience is the branch of neuroscience that studies the neural mechanisms of cognition; that is, it is concerned with understanding how mental processes take place in the brain (Gazzaniga, 1999).[1] Cognitive neuroscience uses methods from cognitive psychology, functional neuroimaging, neuropsychology, and behavioral neuroscience. An influential theoretical approach within cognitive neuroscience is computational neuroscience, which draws on neuroscience, computer science and applied mathematics. It uses mathematical and computational techniques to understand the function of the nervous system (Dayan and Abbott, 2001). In traditional computational neuroscience, the generation of a cognitive theory typically follows the following steps: (1) a set of experiments are run and data on the behavior to be modeled (time, error percentage, etc.) are collected and stored; (2) a neuroscientist generates a cognitive theory of the behavior; (3) the cognitive theory is implemented in the form of a computer program; (4) the computer program is run through the same experiments; and (5), if the output of the computational model is reasonably similar to the values produced by the experiments, the theory is considered valid.

[1] In this paper we use 'cognitive neuroscience' as a generic term covering neuroscience, cognitive science, and cognitive psychology.

The second step, in which a neuroscientist generates a cognitive theory, is a good example of scientific discovery in action. Scientific discovery can be described as heuristic search in combinatorial spaces (Langley et al., 1996; Simon, 1977). In this context combinatorial means that at each point, several decisions are possible, and also that the search space outgrows human capacities to explore it. To avoid these limits, artificial intelligence has developed different search techniques that can devise laws, theories and concepts. These techniques can be used either autonomously or semi-autonomously, and they have been successfully applied in science (Bollobas and Riordan, 1998; Valdes-Perez, 1999). One important class of computational search techniques consists of evolutionary computation, which includes genetic algorithms (Holland, 1992; Mitchell, 1996) and genetic programming (Koza, 1992; Koza, 1994). Genetic Programming (GP) evolves entire computer programs in the form of hierarchical trees using a set of operators and terminals by first generating an initial population of random trees and applying natural selection, crossover, and mutation to breed the following generations. The key of the process is the fitness function used to evaluate the fitness of each program against the desired output. At the end, GP outputs the program that verifies some conditions regarding its fitness value.

GP has become very popular in recent years due to its ability to automatically design complex structures using a tree representation. As evolutionary computation is not as sensitive to local minima and initial conditions as other hill-climbing methods (Koza, 1992), and as it can explore large search spaces efficiently and in parallel, it is ideal in problems where the information is noisy and subject to uncertainty. Evolutionary computation in general and GP in particular have already been used for a wide variety of applications including digital hardware design and optimization (Jackson, 2005), analog hardware design and optimization (Dastidar et al., 2005), solving multi-objective problems (Whigham and Crapper, 2001), design of classifiers (Muni et al., 2004), and also some neuroscientific applications like diagnostic discovery (Kentala et al., 1999), neuromuscular disorders assessment (Pattichis and Schizas, 1996), and interpretation of magnetic-resonance brain images (Sonka et al. 1996).

Considering that the hierarchical tree structure produced by GP is convenient for simulating human mental programs, as we discuss later, and that the generation of cognitive theories is another example of heuristic search within a combinatorial space, we propose to use GP to generate cognitive theories automatically. Our approach uses the experimental data traditionally used to validate theories as a means to compute the fitness of a given model, and thus to control the evolution process. Nevertheless, the automatic generation of cognitive theories using GP faces some problems:

- Lack of standard set of operators. In general, when using GP for typical problems the set of operators used are known, well defined and accepted. In cognitive neuroscience, the set of primitives of the mind, with some minor exceptions, are not known, or generally accepted. This implies uncertainty as to whether the set of primitives used is correct and sufficient to model a given behavior.

- The data used to compute the fitness function have an inherent error due to their

empirical nature, because the original experiments were carried out with humans as subjects[2]. This implies that the operators should also have some error value when being executed, to mimic this non-deterministic behavior. The experimental data usually capture this variability using the standard deviation (*std*) of the results.

- Availability of fitness data. In most previous applications, the desired behavior that is used to evolve GP is well defined and known (for example, when automatically designing hardware, the desired outputs for each input are well defined). This is not the case for cognitive neuroscience, where the information needed to evolve programs is not well defined (for example, different authors can report different results for the same experiment) and is widely dispersed.

In this paper, we present a novel strategy to automatically generate cognitive theories using a combination of GP and experimental data found in the literature. The goal of our work is to help neuroscientists elaborate more complex and veridical explanations of behavior. In order to illustrate our strategy we present a simple application example using a task commonly used in neuroscience, the delayed-match-to-sample (DMTS) task.

The outline of this paper is as follows: In Section 2 we give a brief introduction to Genetic Programming, focusing on how the different characteristics affect its application to the generation of cognitive theories. In Section 3 we present the basic strategy used to automatically evolve cognitive theories. In Section 4 the environment designed and implemented for the evolution of theories is detailed. Section 5 presents an application of our method, using the delayed-match-to-sample task; we also discuss the cognitive theories generated by our approach. The paper finishes with conclusions and a discussion of future work.

2. Introduction to Genetic Programming

GP is an evolutionary computational technique based on reproduction of the fittest that evolves a population of computer programs based on some requirements (Banzhaf et al., 1998; Angeline and Pollack, 1992; Koza, 1994). By operating on variable size digital chromosomes (Mitchell, 1992), GP removes some of the limitations of genetic algorithms, mainly the necessity to use fixed-length chromosomes, the difficulty in representing hierarchical structures, and the lack of dynamic variability (Koza, 1992).

Each individual program in GP is expressed using a hierarchical tree composed of terminals and operators, and has associated a fitness value that indicates the program quality with respect to the goal of the evolution. The evolution process is implemented using three genetic operators (reproduction, crossover and mutation), which control which individuals pass from one generation to the next one. The evolution process in

[2] The same is true for experiments done with animals, for which the concepts presented in this paper also apply.

Table I. Set of variables that define GP evolution [9].

Parameter	Content
N	Population of each generation
M	Maximum number of generations
MDNI	Max depth of new individuals
MDIAC	Max depth of individuals after crossover
FRF	Fitness reproduction fraction
CAPF	Crossover at any point fraction
CAFPF	Crossover at function point fraction
MDNS	Max depth for new subtrees in mutants
MS	Method of selection
MIG	Method of initial generation of the population
MS	Method or combination of methods used to Stop the evolution procedure
RS	Seed used by GP for modelling randomness

GP comprises the following steps:

• Step 1: Selection of admissible set of Operators (*O*), set of Terminals (*T*), fitness function and GP parameters. The set of operators *O* is composed by the operations and functions available to the GP system, and, in traditional applications it contains logical operators or arithmetic and mathematical functions. The set of terminals *T* is composed by the inputs of the system and constants. The set *{O, T}* must have two properties: (1) closure and (2) completeness. Closure requires that the output of any member of *{O, T}* can be the input to any member of *O*, i.e. that there are no restrictions in the combinations of terminals and operators. Completeness requires that *{O, T}* provides enough functionality to present a solution to the problem. When working in environments where *O* is not well established, such as computational neuroscience, the design of these elements can be complex. Regarding the fitness function, it is defined as a mathematical function that obtains the fitness between the program being evaluated and the desired behavior. Typically, this is some form of distance between the output produced by the program and the desired behavior. Regarding the GP parameters, there are a number of parameters that need to be defined before starting the evolution. Table I presents these parameters.

• Step 2: Generation of initial population of trees. Using *{O, T}* a set of *N* trees is randomly generated. These trees can be of different sizes and shapes, and can be generated: (1) fully randomly, (2) as set of initial programs given by the designer, (3) as random variation of plausible programs, or (4) a combination of the previous methods. With fully random generation, there are different techniques: full, grow, and half-and-half (Koza, 1992). In a neuroscience context, the possibility of feeding the system with existing explanations (i.e., programs) can be very useful to obtain more elaborated cognitive theories.

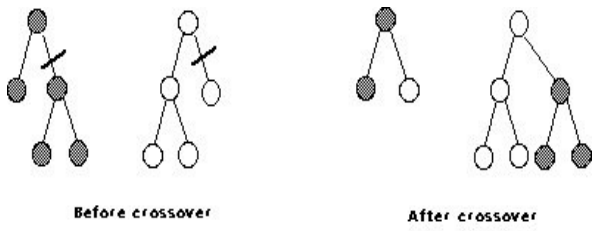

Figure 1. Example of Crossover

• Step 3: Calculation of the fitness of each program. The fitness is computed based on the performance of each program on a set of fitness cases, where both the input and the output are known. There are several measures for fitness typically used: raw fitness, standardized fitness, and normalized fitness, among others. With cognitive programs, the fitness cases consist of behavioral data of humans performing some tasks, where the fit is the amount of variance in the empirical data accounted for by the cognitive program.

• Step 4: Probabilistic application of the genetic functions of selection, crossover, or mutation. The selection of one of the methods is given by the probabilistic values defined in the first step of the process. The selection function replicates one individual in the new population. The most typical selection function is fitness-proportionate, where the probability for an individual to be selected depends on its fitness compared with others in the population. The crossover function refers to producing two offspring from a random point in each of the two parents and swapping the resultant subtrees. Fig. 1 presents an example of crossover. The two programs used for crossover are chosen according to their fitness, and the two resulting programs are passed onto the new generation. The mutation function consists in replacing a subtree below a random point by a randomly created subtree. The mutated solution is passed onto the new generation. Mutation helps to maintain diversity.

• Step 5: Repetition of the process. Once we have the new generation, steps three and four are repeated until a solution is found. GP does not necessarily converge, so, in general, the process stops once the number of maximum generations has been reached or a program with fitness value smaller that a predefined value is produced.

For the implementation of the system and the experiments described in this paper, we have used the standard Lisp implementation of GP detailed by Koza (1992). GP for modeling theories of cognitive behavior faces some limitations:

• (1) Tree Structure. In general, the basic hierarchical tree structure is enough for representing the programs needed, but in some cases more complex structures are needed; in that case, several techniques exist to implement cyclic or recursive function calls, such as automatically defined functions (Koza, 1994; Angeline and Pollack, 1992). Regarding the representation of cognitive theories, it is assumed that simple systems, both natural and artificial, are more likely to evolve into complex systems if they are organized as modules and hierarchies (Simon, 1992). Hierarchical organizations have been proposed for brain structures (Churchland and Sejnowski, 1992), cognitive processes (Kosslyn and Koenig, 1992) and knowledge representations (Gobet, 2001). While the assumption of modularity has sometimes been disputed (Elman et al., 1996), it can be said that it is accepted by mainstream neuroscience (Shallice, 1990). This implies that the hierarchical tree structure may be considered sufficient to model cognitive behavior models.

• (2) Bloating. Bloating is an inherent GP problem and represents the accelerated growth of the trees in successive generations (Langdon and Poli, 1998; Lones and Tyrrell, 2002). This problem also relates to the execution time problem, because larger programs consume more resources. It also has other consequences, especially to contribute towards the overfitting of the solution (Burke et al., 2004). This problem is of special relevance with modeling cognitive behavior, because as a general rule simpler and smaller behavior explanations are considered better solutions. Nevertheless, it also has some positive aspects; mainly the fact that because the brain is a highly redundant structure, this redundancy can be introduced by the bloating of the structures generated. In our context bloating is to some extent a good property that needs to be controlled to avoid overfitting of the solution and control the execution time of the cognitive theories generated. A possible solution to this problem is to limit the depth of the trees generated.

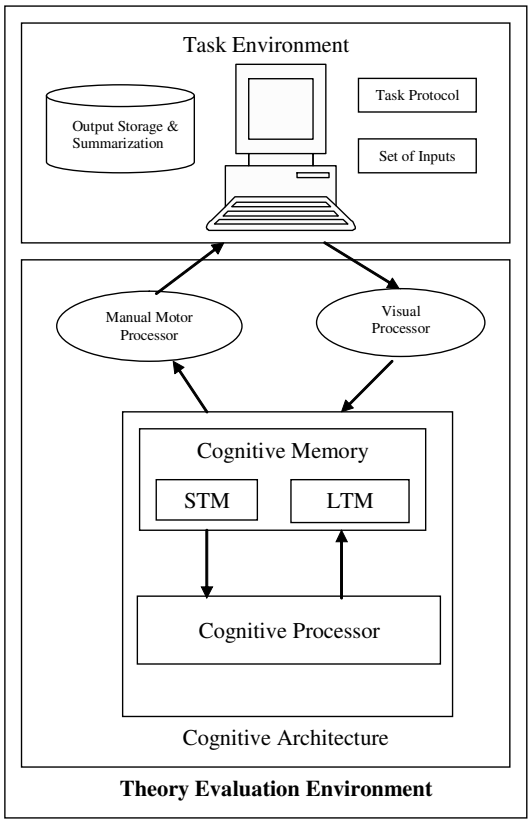

Figure 2. Basic Cognitive Architecture and Task Environment.

3. Strategy for Automatic Generation of Cognitive Theories

In order to generate cognitive theories of behavior, an environment that evaluates theories needs to be defined. The Theory Evaluation Environment (TEE), presented in Fig. 2, is composed by two elements: (1) a Cognitive Architecture and (2) a Task Environment. TEE is designed to evaluate a behavior theory implemented in the Cognitive Architecture for an experiment presented by the Task Environment.

The Cognitive Architecture defines the elements and connections that take place in the human brain. Designing a cognitive architecture is an open research field with different approaches, mainly symbolic (Anderson et al., 2004) and connectionist (O'Reilly, 1998). In our approach, we consider a very basic cognitive architecture that only presents the main blocks taking part in the execution of mental programs in humans. The Cognitive Architecture presents four main elements: (1) Visual Processor,

(2) Cognitive Memory, (3) Cognitive Processor, and (4) Manual Motor Processor.

- *Visual Processor*. The Visual Processor comprises all the mechanisms necessary to generate an image and the mental image that is produced after "seeing" an object.
- *Cognitive Memory*. The Cognitive Memory groups the components and processes that store information, including their interactions. Typically two elements are defined, Long-Term Memory (LTM) and Short-Term Memory (STM). The literature presents a considerable number of studies about their definition, interactions, and learning processes (Cowan, 2001; Anderson, 1983; Eichenbaum, 2002).
- *Cognitive Processor*. The Cognitive Processor runs the cognitive program using data from the Cognitive Memory.
- *Manual Motor Processor*. The Manual Motor Processor, which receives the input from the Cognitive Processor, controls the manual actions taken by the individual running the task.

The Task Environment is defined as the set of elements necessary to define an experiment. This includes not only material elements (screen, keyboard, stimuli used by the experiments, etc.) but also the description and the protocol of the experiment. The Task Environment also summarizes the outputs produced for each input in order to present them in a compact way, typically by presenting the mean and standard deviation (*std*).

The stages of our approach for automatically generating cognitive theories are:

- Step 1: *Define set of operators O and Terminals T, GP parameters, and Fitness Function*. The experiment implemented in the Task Environment will define which kind of operators are needed. These operators (such as access to STM, inhibition of visual information, or matching two visual pieces of information) are to some extent documented in the literature. The set of terminals basically specify the inputs of the

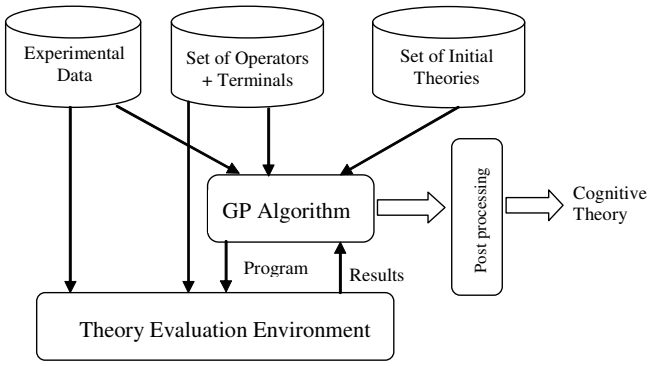

Figure 3. Environment for automatic generation of cognitive theories.

experiment, and if needed, the set of constants used. Regarding GP parameters and the fitness function, typical values can be used, although some experimentation can be useful to obtain better solutions.

• Step 2: *Codification of current knowledge of the cognitive task being modeled.* The translation of current theories into formal programs using the set of predefined operators *O* and terminals *T* (defined on the first step) might produce a large number of different programs, which might reveal inconsistencies across authors or even within authors. These programs can be used as the initial seed of population and also for testing that the set of operators *O* and terminals *T* are sufficient for carrying out the task.

• Step 3: *Construction of a database containing empirical results of the task being modeled.* Testing the fitness of the theories requires the creation of a database of results from human studies of the behavior being modeled. It is then possible to compute the fitness of a given theory by comparing the predictions of the theory with the empirical data. The database will contain, among others, the description of the elements used in the experiments, the set of inputs used and the results obtained in each case. A formal description of the database is presented later.

• Step 4: *Use of genetic-programming techniques to evolve the theories.* The final stage uses evolutionary computation to optimize the search through the spaces of programs. GP receives as inputs: (1) the database of experimental data, which will be used to test the fitness of each program, (2) the set of primitives, which will be used to evolve behavior models, and (3) the set of initial behavior theories, that can be used as seeds to evolve more refined theories. Fig. 3 presents the interconnections between these elements. TEE receives each program generated by GP, executes the experiments defined in the Task Environment using the received program as part of the Cognitive Processor, collects the empirical results, and sends them to GP for fitness evaluation.

• Step 5: *Post-processing of the generated theories*. Theories generated by GP will in general contain branches that are not relevant, functions that repeat actions, or functions that cancel the action of each other. In order to better present and study the behavior of a theory, the generated theories are simplified. The simplification rules are dependant of the task and the set of operators.

$$\mathbf{Experiments} = (\mathbf{Experiment}_1 ... \mathbf{Experiment}_m)$$

$$\mathbf{Experiment}_i = (\mathbf{Prototype}_i\ \mathbf{Protocol}_i\ (\mathbf{Input}_{i,1} ... \mathbf{Input}_{i,n(i)})\ (Output_{i,1} ... Output_{i,n(i)})\ \mathbf{Values}_i), i = 1,...,m$$

$$\mathbf{Prototype} = ((NameInput_{i,1}\ ...\ NameInput_{i,J(i)})(OutputName_{i,1} ... OutputName_{i,o(i)}))$$

$$\mathbf{Protocol}_i = (((NameInput_{i,1} ... NameInput_{i,z} \mid IDLE) ... (NameInput_{i,1} ... NameInput_{i,f} \mid IDLE))$$

$$\mathbf{ExposureTime}_i)$$

$$\mathbf{ExposureTime}_i = (t_1 .. t_{s(i)})$$

$$\mathbf{Input}_{i,j} = (Stimulus_{i,j,1} ... Stimulus_{i,j,J(i)}), i = 1,...,m,\ j = 1,...,n(i)$$

$$\mathbf{Values}_i = (NumberOfSubjects_i\ \mathbf{Results}_i)$$

$$\mathbf{Results}_i = (Accuracy_i\ stdAccuracy_i)$$

Figure 4. Formal definition of the Experimental Data Database.

$$\mathbf{O} = \{\mathbf{Operator}_1 ... \mathbf{Operator}_o\}$$

$$\mathbf{Operator}_i = (\mathbf{Prototype}_i\ \mathbf{OperatorDefinition}_i\ \text{ExecutionError}_i), i = 1,...,o$$

$$\mathbf{Prototype}_i = ((InputName_{i,1} ... InputName_{i,n(i)})(OutputName_{i,1} ... OutputName_{i,o(i)}))$$

$$\mathbf{OperatorDefinition}_i = (\text{LISP code})$$

$$\text{ExecutionError}_i \in [0,1]$$

$$\mathrm{T} = \{\mathrm{Terminal}_1 ... \mathrm{Terminal}_r\}$$

Figure 5. Formal definition of the Set of Operators Database.

Due to the inherent fuzziness of human behavior, this proposed approach faces some problems, which are presented in the following section.

4. Environment for automatic generation of cognitive theories

The environment, presented in Fig. 3, comprises three main elements: (1) Database of Experimental Data, (2) Set of Operators and Terminals, and (3) Theory Evaluation Environment.

4.1 Database of Experimental Data

The database of experimental data contains all the relevant information regarding the experiments for which a behavior model is going to be evolved. The information for each experiment includes: the name of the input variables, the name of the output variables, the communication protocol between the Task Environment and the Cognitive Architecture, the set of inputs given to the system, the set of outputs obtained for each input, the number of subjects used for the experiments, the number of trials per subject, and a vector of results, which typically include mean and standard deviation (*std*) of accuracy.

Fig. 4 presents a formal representation of the structure of the cognitive database,

```
Function Operator_i (Inputs)
    Obtain Error using ExecutionError_i
    If Error
        return (random_valid_value)
    else
        Operator_i.OperatorDefinition(Inputs)
        return (Output)
    end_if
end_Function
```

Figure 6. Pseudo-code for a generic Operator.

with *m* the number of experiments described, *l(i)* the dimension of experiment *i* (i.e. its number of inputs), with *i=1,…,m*, *NumberofSubjects* the number of subjects that took part in each experiment, and *n(i)* the number of trials of each subject. The protocol is described by the order in which the input variables are presented and their exposure time. This database provides the information needed by the Task Environment of TEE to define its parameters, and by the GP algorithm to calculate the fitness function.

Creating such a database for a specific experiment can be very complex, because the description of the experiments in the neuroscience literature does not necessarily contain the required information. Section V presents an example of the database we created for the delayed-match-to-sample task, a typical neuroscience task both with animals and humans.

4.2 Set of Operators (O) and Terminals (T)

The set of operators and terminals can also be expressed in the form of a database. The set of operators needed to solve a task can be present to some extent in the existing literature. When designing this set of operators *O,* closure and completeness properties have to be verified. Regarding the set of terminals *T*, in general they are the names of the set of input variables of the task.

Fig. 5 presents the formal definition of *O* and *T*, where *o* is the number of operators and *t* is the number of terminals. Each operator contains: (1) a prototype that defines the number and name of inputs and output, (2) the implementation of the operator in Lisp, and (3) the execution error. The execution error is a very important parameter, and its goal is to model the inherent fuzziness of empirical data with humans. Each operator has an *ExecutionError* parameter that indicates the probability that this operator is incorrectly executed. This parameter enables us to model human errors due to imperfections in the motor processor, distractions from the environment, or incomplete understanding of the instructions. When an operator is executed with error, it outputs a random but valid value. Fig. 6 presents the basic pseudo-code for implementing a generic operator. We have not found references of possible error rates of

Input: Program, *Experiments*, *O*
Output: $Accuracy_i, StdAccuracy_i, i = 1,..,m$

For each $Experiment_i, i = 1,...,m$
  Load $Protocol_i$
  For each $S = 1$ *to* $NumberofSubjects_i$
    $NumberOfCorrectSolutions_S = 0$
    $TotalExecutionTime_S = 0$
    For $L = 1$ to $n(i)$
      $OutputE$=Execute(Program, $\mathbf{Input}_{i,L}, O, Protocol$)
      If $OutputE = Output_{i,L}$
        $NumberOfCorrectSolutions_S = NumberofCorrectSolutions_S + 1$
      End_If
    End_for
  End_for
  $Accuracy_i$=mean($NumberOfCorrectSolutions_1,...,NumberOfCOrrectSolutions_{NumberOfSubjects}$)
  $StdAccuracy_i$=std($NumberOfCorrectSolutions_1,...,NumberOfCOrrectSolutions_{NumberOfSubjects}$)
End_for
return($Accuracy_1, StdAccuracy_1,..., Accuracy_m, StdAccuracy_m$)

Figure 7. Pseudo-code for the Theory Evaluation Environment.

basic neuroscience operators. Not all operators should have the same execution error, it is sensible for example to assign and execution error of 0 to infrastructural operators. In Fig. 3, TEE access this database to obtain the definition of the operators in order to execute the programs evolved and GP to access the set of *O* and *T*.

4.3 Theory Evaluation Environment

The Theory Evaluation Environment (TEE), which has been implemented in Lisp, produces, for a given program and a given experiment, the values that describe the results of the experiment. In neuroscience, the output values of a theory are quite standard, and, for example, describe to which extent the model carries out a task correctly (*Accuracy*) or the variability of correct performance (*StdAccuracy*). The output of TEE is returned to the GP environment which will use this value to obtain the fitness of that specific program by comparing the values received with the experimental data contained in the experimental database.

Fig. 7 presents the basic pseudocode implemented by TEE. For each one of the experiments of the database *i=1,…,m*, TEE runs for each subject, for *S=1,…,NumberOfSubjects*, and for each input the program, for *L=1,…,n(i)*, and checks if the solution provided is the same one as the one contained in the experimental database. Once the loops have been executed, the system calculates a mathematical description of the behavior suitable to be used for evolution purposes. The key function of the process is the "Execute" function. When this function is given a theory in the form of a program, a set of input values, and *O* (which provides the Lisp code for the primitives), it obtains the output of the system. The execution of the program is

easily done by implementing it as a S-expression in Lisp.

In TEE the problem of execution time, present in any GP problem, is made worse by the number of loops that have to be run to evaluate the fitness of each program.

5. Automatic Generation of Delayed-Match to Sample (DMTS) Cognitive Theories

This section implements the previous ideas for a well-known neuroscience task, DMTS. It also compares how a traditional approach of generating theories will work, and shows how the GP approach can help neuroscientists generate cognitive theories.

5.1 Experiment Description

As an example of the previous ideas we have considered the delayed match to sample (DMTS) task. In this task a stimulus is first presented for a given amount of time, followed by a delay. Then, two stimuli are presented, and the task is to select which of these two stimuli matches the stimulus presented first, where one of them always matches the first one. Fig. 8 presents an example of the chain of stimuli presented by DMTS with pictures of tools as stimuli. The experiment usually takes place on a computer, so that the subject receives the stimuli through the screen and the outputs of the subject are received using the computer keyboard. A substantial number of studies

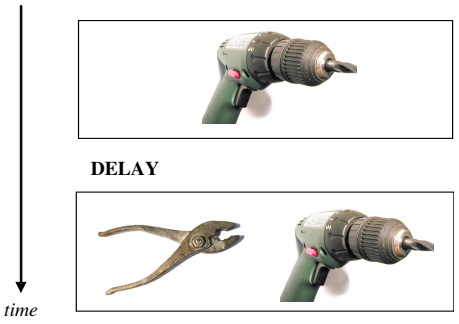

Figure 8. Example of stimuli presented by DMTS when using "tools" as the generic type. Photographs courtesy of www.freeimages.co.uk.

focus on this task (e.g., Chao et al, 1999; Elliot and Dollan, 1999; Grady et al., 1998; Habeck et al., 2003; Mecklinger and Pfeifer, 1996; Zubicaray et al., 2001), ranging from comparison of results under different conditions to identifying which areas of

the brain take part in the process.

In this example, we used the data of Chao, Haxby and Martin (1999). Their paper focuses on the areas of the brain that are activated while executing a DMTS task using different conditions, while at the same time providing results for mean and standard deviation of accuracy for each task. The conditions on which the authors focused were: (1) pictures of animals and (2) pictures of tools. The first element is presented for 1 second in the center of the screen, then there is a delay of 0.5 seconds, after which the two elements are presented for 2 seconds, one on the left and the other on the right of the screen. During these two seconds, the individual participating in the experiment has to select which of the two elements, the one on the right or the one in the left, is the same as the first element presented. A failure to respond within those two seconds is considered as an incorrect answer. The task was run for each condition by four subjects, each one running 60 trials. The paper gave no details about which exact pictures of animals and tools were used. With animals, the mean percentage of correct answers was 97% with an *std* of 1.4%, and with tools, it was 95%, with an *std* of 1.2%. Here, we are not interested in the difference between the two conditions (which were not statistically significant) but in the absolute values of the mean and standard deviations.

The reason for choosing this study is the simplicity of the experiment. This allows us to make some very relevant assumptions:

- The experiment only deals with known elements (animals and tools). This characteristic makes it possible to focus on the process of comparing elements more that on the process of actually learning the elements of the experiment. We assume that all the elements presented to a subject during the experiment are known to that subject (i.e., they are already in LTM), which implies that no learning takes place. The process of accessing LTM can thus be waived; the subject will always find the elements and put them in STM.
- The whole process, for a given set of inputs, takes place in 3.5 seconds. This allows another important assumption: STM can retain information perfectly within this period of time. Hence, no modeling of STM decay is needed.

5.2 Traditional Computational Neuroscience Approach

In this section we generate a behavior model for DMTS in a traditional way. In order to facilitate the comparison with a GP-generated theory, we also present the human-generated theory in the form of a program. First, we start by defining the cognitive memory: we assume that no LTM is needed and that STM works as a queue with four elements (Cowen, 2001; Gobet and Clarkson, 2004). Regarding the program of the cognitive processor our theory consists in the following steps:

Step 1) The first input, called *I1* and presented in the center of the screen, is processed by the Visual Processor and stored in STM.
Step 2) During the delay, no action is taken.

Step 3) The second input, called *I2* and presented in the left part of the screen, is processed by the Visual Processor and stored in STM.
Step 4) The third input, called *I3* and presented in the right part of the screen, is processed by the Visual Processor and stored in STM.
Step 5) During the 2 seconds that *I2* and *I3* are presented in the screen, the following process takes place: If *I1* is equal to *I2*, the same element is the one the left, and if not, the same element is the one on the right. Considering that STM works as a queue, we are actually comparing the third position of the queue (where *I1* is stored) with the second position (where *I2* is stored).
Step 6) The result of the comparison is passed to the Manual Motor Processor, which selects "left" or "right" accordingly.

Using a traditional computational neuroscience approach, the previous theory first needs to be expressed in the form of a program, and, then be validated by comparing its output (mean and standard deviation of accuracy) to the experimental data reported by the authors. In order to present the previous DMTS theory in the form of a tree, we need to define the inputs and the functions of the program (something that actually will be very useful later to define *O* and *T*). The theory has three inputs, *I1*, *I2* and *I3*, and one output with two possible values: "left" or "right". Regarding the functions, it is clear from the previous steps that we need an operator to write in STM (*WSTM*), and another one, called *AdvancedCompare*, that compares the third and the second position of STM and outputs "left" or "right" depending on the result of the comparison. In order to be able to express the program in the form of a tree, we also need an operation to express sequentiality, called *Sequence,* which outputs the value received from the last parameter executed. The program can be expressed as:

(sequence (sequence (WSTM I1) (WSTM I2) (WSTM I3)) AdvanceCompare) (1)

Fig. 9 graphically presents the program tree that expresses our theory, where squares indicate inputs, ovals indicate operators or functions, and arrows indicate the flow of information. In order to check the validity of our human-generated theory, the program of Fig. 9, in the form of a Lisp S-expression, was evaluated in TEE.

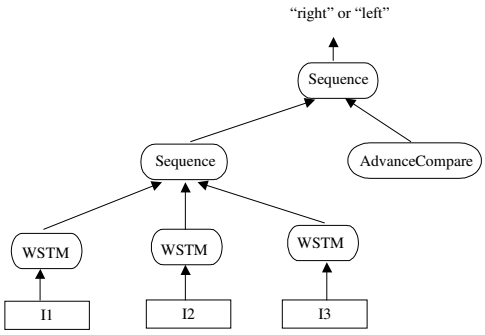

Figure 9. Tree implementation of the human-generated DMTS theory.

Table II. Set of operators defined for *O1*.

Operator	Description
Progn2	**Function**: executes two inputs sequentially. **Input:** Input1, Input2. **Output:** The output produced by Input2.
putSTM	**Function:** Writes the input in STM. **Input:** Input1. **Output:** The element written in STM (Input1).
Compare12	**Function:** Compares positions 1 and 2 of STM and returns NIL if they are not equal or the element if they are equal. **Input:** None. **Output:** NIL or the element being compared.
Compare13	As Compare 12 but with elements 1 and 3 of the STM.
Compare23	As Compare 12 but with elements 2 and 3 of the STM.

Table III. Set of operators defined for *O2*.

Operator	Description
Progn2	**Function**: executes two inputs sequentially. **Input:** Input1, Input2. **Output:** The output produced by Input2.
putSTM	**Function:** Writes the input parameter in STM. **Input:** Input1. **Output:** The element written in STM.
Compare12	**Function:** Compares elements 1 and 2 of STM and returns NIL or TRUE. **Input:** None. **Output:** NIL or TRUE.
Compare13	As Compare12 but with elements 1 and 3 of the STM.
Compare23	As Compare12 but with elements 2 and 3 of the STM.
OpNIL	**Function:** Produces and returns NIL **Input:** None **Output:** NIL
AccessSTM1	**Function:** Reads the element 1 of STM and outputs it. **Input:** None **Output:** Value contained in position 1 of STM
AccessSTM2	As *AccessSTM1* but with position 2.
AccessSTM3	As *AccessSTM1* but with position 3.
If_condition	**Function:** Evaluates a condition and executes input2 if it is TRUE or INPUT3 if it is NIL. **Input:** Condition, Input2, Input3 **Output:** The value produced by Input2 or input3 depending on the value of condition.

Each function was assigned an error factor (*ExecutionError*) of 0.02%, and in order to have reliable data we executed the experiment and collected the results ten times. This gave us a total of ten values for *Accuracy* and *StdAccuracy*, which averaged to 95.7% and 1.8% for animals, and 96.6% and 1.38% for tools. When compared with the experimental values of 97% and 1.4% for animals and 95% and 1.2% for tools, our human-generated theory explains at an acceptable level the cognitive process under-

Table IV. Parameters used for evolution.

Parameter	Value
MDNI	*10*
MDIAC	*12*
FRF	*0.1*
CAPF	*0.2*
CAFPF	*0.2*
MDNS	*1*
MS	*Fitness proportionate*
MIG	*Ramped half and half*
RS	*0.8*

taken by humans to solve the DMTS task. The next section solves the same problem but using the GP approach.

5.3 GP Theory Generation Approach

This section follows the strategy defined in Section III to automatically generate theories that solve DMTS. As in our example of a theory generated by the traditional computational neuroscience approach, there is no LTM, and STM is a four-element queue.

5.3.1 Definition of Operators (O), Terminals(T), GP parameters and Fitness Function

When defining the set of operators *O*, several factors need to be taken into account: (1) the cognitive architecture, (2) existing theories found in the literature, and (3) the definition of the task to be modeled. Also *O* should verify completeness and closure. In general, in an environment such as neuroscience, completeness can only be established via experimentation. By contrast, the problem of closure can be solved when designing the operators. In our case, we have designed all primitives to have as inputs and outputs either "False" (the Lisp value *NIL*) or one of the values of the inputs. In DMTS, the simulation has to output "right" of "left" depending on which element is equal to the first one, and in our case this has been solved by using "False" to indicate "left" and any element of the experiment (not false) to indicate "right".

Considering the previous factors we have defined two sets of operators, *O1* and *O2*. Table II and Table III present the operators defined. In both cases all the operators have been assigned an execution error (*ExecutionError*) of 0.02. The two sets of operations represent very different philosophies about how to model DMTS. While *O1* is a high level approach with complex operators, like *Compare*, that actually contains a condition and an access to STM in it, *O2* has a lower granularity with simpler operators. Both approaches have their advantages and disadvantages: *O2* will generate

models of behavior with smaller granularity, which will be more difficult to understand, and will define a higher dimensional search space, but it also makes it possible to design operators that can be used to model other tasks. *O1* produces a smaller dimensional space, which implies that solutions could be found in a smaller amount of time, and because of the high granularity the behavior of the models will be easier to understand. Nevertheless, operations as complex as *Compare* in *O1* will likely be useful only for DMTS, which reduces the applicability of the results obtained.

Regarding the set of terminals *T* for both sets of operators, in this case only three variables (*I1*, *I2* and *I3*) and no constants are needed. Regarding the fitness function, we have decided to use a very simple measure based on the distance between the elements that describe the desired behavior and the behavior of the program being evaluated. With $(Accuracy_1\ stdAccuracy_1\ Accuracy_2\ stdAccuracy_2)$ being the desired behavior expressed by Chao et al. (1999) for faces (experiment 1) and tools (experiment 2), with values *(0.95 1.4 0.97 1.4)*, and $(GPAccuracy_1\ GPstdAccuracy_1\ GPAccuracy_2\ GPstdAccuracy_2)$ the vector returned by TEE after evaluating a theory in both experiments, the standardized fitness *F* is defined as:

$$F = \sum_{j=1}^{2} \left\| Accuracy_j - GPAccuracy_j \right\| + \sum_{j=1}^{2} \left\| StdAccuracy_j - GPStdAccuracy_j \right\| \tag{2}$$

Table IV presents the parameters used for the GP evolution; *M* and *N* are tested with different values. The rest of values, unless noted, are constant for all the evolutions. The Method of Stop (*MS*) is one of the following: either (1) *M* reaches its limit value, or (2) the following two conditions are verified at the same time:

$$\sum_{j=1}^{2} \left\| Accuracy_j - GPAccuracy_j \right\| < 0.1 \tag{3}$$

$$\sum_{j=1}^{2} \left\| StdAccuracy_j - GPStdAccuracy_j \right\| < 0.05 \tag{4}$$

This implies that we consider values for accuracy acceptable if the difference is smaller that 10% for the means and smaller that 5% for *std*. We consider that a solution has been found only if GP stops because (3) and (4) are verified.

$$\textbf{Experiments} = (\textbf{Experiment}_{\text{Animals}}, \textbf{Experiment}_{\text{Tools}})$$

$$\textbf{Experiment}_{\text{Animals}} = (\textbf{Prototype}_{\text{Animals}}\ \textbf{Protocol}_{\text{Animals}}\ (\textbf{Input}_{i,1}...\textbf{Input}_{i,60})\ (\textit{Output}_{i,1}...\textit{Output}_{i,60})\ \textbf{Values}_{\text{Animals}})$$

$$\textbf{Prototype}_{\text{Animals}} = ((\textit{I1 I2 I3})(\textit{O1}))$$

$$\textbf{Protocol}_{\text{Animals}} = \textit{((I1)(IDLE)(I2 I3))(10.5 2))}$$

$$\textbf{Values}_{\text{Animals}} = (4\ (0.97\ 1.4\ ))$$

$$\textbf{Experiment}_{\text{Tools}} = (\textbf{Prototype}_{\text{Tools}}\ \textbf{Protocol}_{\text{Tools}}\ (\textbf{Input}_{i,1}...\textbf{Input}_{i,60})\ (\textit{Output}_{i,1}...\textit{Output}_{i,60})\ \textbf{Values}_{\text{Tools}})$$

$$\textbf{Prototype}_{\text{Tools}} = ((\textit{I1 I2 I3})(\textit{O1}))$$

$$\textbf{Protocol}_{\text{Tools}} = \textit{((I1)(IDLE)(I2 I3))(10.5 2))}$$

$$\textbf{Values}_{\text{Tools}} = (4\ (0.95\ 1.2\ ))$$

Figure 10. Experimental Database for DMTS task.

5.3.2 Construction of a Database of Experimental Data

In this step, a database containing the experimental data described in (Chao et al., 1999) has been constructed. Fig. 10 presents an instantiation of the data structure defined in Fig. 4 that contains the experimental data needed for the experiment with faces and tools. It contains the protocol and the time exposures, which are the same in both cases, the number of trials, 60, the number of subjects, 4, and the values obtained in each experiment. Inputs and Outputs have not been detailed for space purposes, but they were generated using a pseudo-random sequences of digits.

5.3.3 GP-generated Cognitive Theories for DMTS: Analysis of Results

This section details the solutions obtained when using GP with the experimental data for the DMTS task. The theories presented have already been postprocessed to better present their behavior. The environment produced a variety of solutions when using *O1* and *O2*. Both sets of operators were executed ninety times, with *N=20, 40, 60* and *M=50, 80, 100* and for ten values of RS (the random seed used by GP). Fig. 11 presents the theory generated, expressed in the form of a tree, when using *O1* and *N=20* and *M=50*. Fig. 12 presents the theory produced when using *O1* and *N=60* and *M=80*. Fig. 13 is the theory produced with *O1* and *N=40* and *M=80*, and Fig. 14 with *O2* and *N=60* and *M=80*. Other combinations of *N* and *M* did not produce any solution. As can be seen, any of the theories generated basically represent the same behavior: writing the inputs in STM and then comparing them.

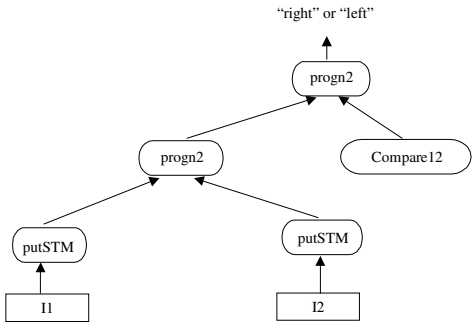

Figure 11. Tree of the GP generated theory with *O1*, *N=20* and *M=50*.

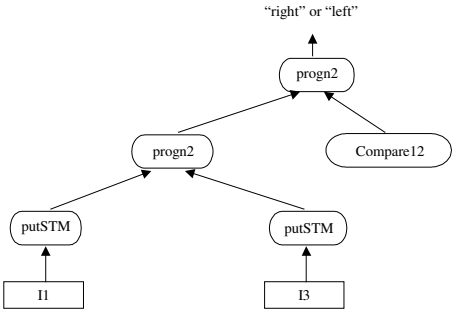

Figure 12. Tree of the GP generated theory with *O1*, *N=60* and *M=80*.

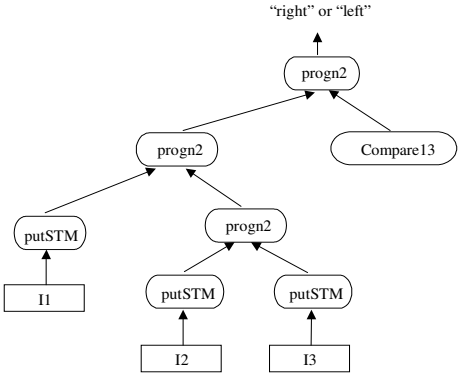

Figure 13. Tree of the GP generated theory with *O1*, *N=40* and *M=80*.

The goal of our system is to help neuroscientists find some interesting characteristics of the behavior that is being modeled, which would be far more difficult if a traditional approach was used. For example, in our case, the GP approach enabled to find that it is not necessary to write the three inputs in STM, but that it is enough to encode *I1* and one of the other two inputs, *I2* or *I3* (see the theories presented in Fig. 11 and

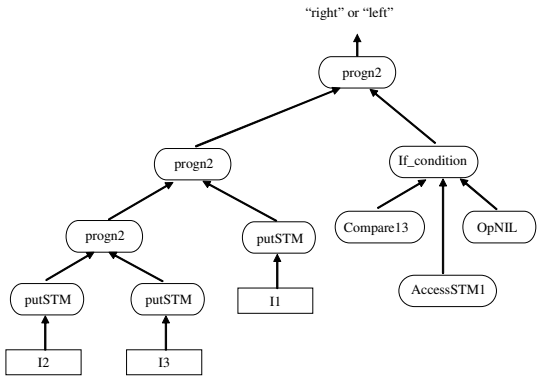

Figure 14. Tree of the GP generated theory with *O2*, *N=60* and *M=80*.

Fig. 12). That is, if one element is not the same as the original, the other must be equal to it. Thus, in order to solve the problem when using *O1*, the minimum set of operators is *{progn2, putSTM}* and one of the operators *Compare12* or *Compare13*, the other two operators not being necessary. Some readers of a previous version of this paper found this explanation counter-intuitive. Surprising and counter-intuitive explanations are not necessarily a bad thing in science, and they can indeed lead to important discoveries (Simon, 1977). While we do not know any empirical data that directly support the proposed GP-theories, we do not know of any study that directly refutes this explanation either. Thus, an unexpected if modest contribution of this paper is to propose a hypothesis about behavior in the DMTS task that can be tested by further empirical studies.

The solution obtained with *O2*, although basically producing the same behavior as the one obtained with *O1*, produces a more complex tree because of the lower granularity of its operators. When using *O2*, the lower granularity of the operators affects the dimensionality of the search space and the density of the solutions. As a result, only one in ninety evolutions of *O2* produced a solution, compared to three in ninety when using *O1*. Although having operators with lower granularity has its advantages, these results also show that it is important to balance the level of granularity with the complexity of the search space.

6. Conclusions and Future Work

Cognitive neuroscience has typically used a traditional scientific research approach in which neuroscientists generate a theory and use experimental data to validate it. With the introduction of computational neuroscience, computers have been used to model data and help develop theories. In this paper, we have presented an environment to automatically generate cognitive theories by using GP, where the evolution process was guided by the experimental data. Our approach had three characteristics that differentiate it from traditional GP applications: (1) lack of standard operators, (2)

construction of experimental databases, and (3) simulation of human behavior. We have designed an environment and a strategy for generating cognitive theories and have applied it to a typical neuroscience task, the delayed-match-to-sample task. Our results show that the system can automatically generate cognitive theories, and that these theories can help neuroscientists in the process of understanding how the mind works.

We acknowledge that the task used in this paper (the DMTS task) is very simple and that we used few data points for computing fitness. Thus, it is an open question as to whether our technique applies to more complex tasks. While our study can be seen as an existence proof, there are obviously important issues that must be answered before being confident of the generality of our methodology. However, establishing that our technique is successful with simple tasks is an important first step. We also note that some of the generated theories are counter-intuitive and thus scientifically interesting, leading to predictions that can be tested empirically. Finally, our approach raises difficult technical and conceptual issues, including the need for heuristics filtering the generated theories and methods enabling humans to make sense of these theories.

We plan to apply our approach to generate cognitive theories to other typical neuroscience tasks, especially tasks involving learning, in order to further establish the benefits of using an automatic approach. The inclusion of additional information, such as the response time needed to make a decision, would help select better theories. Neuroscience is not only interested in explaining behavior but also in identifying in which areas of the brain cognitive operators are executed. Actually, the best part of experimental data found in the literature for a given task focus on this topic, more than in generating cognitive theories. We think that by widening our strategy and our environment to include information about the localization of brain activity for a given task, we will be able to evolve not only behavior theories, but also the mapping between these primitives and brain structures (Gobet and Parker, 2005).

ACKNOWLEDGMENT

We thank Guillermo Campitelli for providing advice on the delayed-match-to-sample task, as well as Veronica Dark and anonymous referees for useful comments.

REFERENCES

Anderson, J. R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C. and Qin, Y.L. (2004), 'An integrated theory of the mind', *Psychological Review* 111, pp. 1036-1060.

Anderson, J.R. (1983), 'Retrieval of long-term memory information', *Science* 220, pp. 25-30.

Angeline P.J. and Pollack, J.B., (1992),'The evolutionary induction of subroutines', in *Proc. Of the fourteenth Annual Conference of the Cognitive Science Society*.

Banzhaf, W., Nordin, P., Keller, R.E. and Francone, F.D., (1998), *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs,* New York: PWS.

Bollobas, B. and Riordan, O., (1998), 'On some conjectures of Graffiti', *Discrete Mathematics 179*, pp. 223-230.

Burke, E.K., Gustafson, S. and Kendall, G. (2004), 'Diversity in genetic programming: An analysis of measures and correlation with fitness', *IEEE Transactions on Evolutionary Computation*, Vol. 8, pp. 47-62.

Chao, L.L., Haxby, J.V. and Martin, A. (1999),'Attribute-based neural substrates in temporal cortex for perceiving and knowing about objects', *Nature Neuroscience*, 2, pp. 913-920.

Churchland, P.S. and Sejnowski, T.J. (1992), *The Computational Brain*, MIT Press.

Cowan, N. (2002), 'The magical number 4 in short-term memory: A reconsideration of mental storage capacity', *Behavioral and Brain Sciences, 24*, pp. 87-114.

Cowen, N. (2001), 'The magical number 4 in short-term memory: A reconsideration of mental capacity', *Behavioral and Brain Sciences 24*, pp. 87-114.

Dastidar, T.R., Chakrabarti, P.P. and Ray, P. (2005),'A Synthesis system for analog circuits based on evolutionary search and topological reuse', *IEEE Transactions on evolutionary computation*, Vol. 9, No. 2, pp. 211-224.

Dayan, P. and Abbott, L.F. (2001), *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, New York: MIT Press.

Eichenbaum, H. (2002), *The cognitive neuroscience of memory*, Oxford University Press.

Elliott, R. and Dollan, R.J. (1999),'Differential Neural Responses during Performance of Matching and Nonmatching to Sample Tasks at Two Delay Intervals', *The Journal of Neuroscience*, 19, pp. 5066-5073.

Elman, J.L., Bates, E.A., Johnson, M.H. , Karmiloff-Smith, A., Parisi, D. and Plunkett, K. (1996), *Rethinking innateness, A connectionist perspective on development*, MIT Press.

Gazzaniga, M.S. (1999), *Conversations in the Cognitive Neurosciences*. New York: MIT Press.

Gobet, F. (2001), 'Is experts' knowledge modular?', in *Proc. of the 23rd Meeting of the Cognitive Science Society*, pp. 336-431.

Gobet, F. and Clarkson, G. (2004), 'Chunks in expert memory: Evidence for the magical number four …or is it two?', *Memory* 12, pp. 732-747.

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

Grady, C. L., McIntosh, A.R., Bookstein, F., Horwitz, B., Rapoport, S.I. and Haxby, J.V. (1998), 'Age-related changes in regional cerebral blood flow during working memory for faces', *NeuroImage 8*, pp. 409–425.

Habeck, C., Hilton, J., Zarahn, E., Flynn, J., Moeller, J.R. and Stern, Y. (2003), '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*, pp. 1723– 1733.

Holland, J.H. (1992), *Adaptation in natural and artificial systems*, Cambridge: MIT Press.

Jackson, D. (2005), 'Evolution of processor microcode', *IEEE Transactions on Evolutionary Computation*, Vol. 9., No. 1, pp. 44-59.

Kentala, E., Laurikkala, J.,Pyykko, I. and Juhola, M., (1999),'Discovering diagnostic rules from a neurologic database with genetic algorithms', *Annals of Otology, Rhinology and Laryngology* 108, pp. 948-954.

Kosslyn, S.M. and Koenig, O. (1992), *Wet Mind*, New York Free Press.

Koza, J. (1992), *Genetic Programming: On the programming of computers by means of natural selection*, MIT Press.

Koza, J. (1994), *Genetic Programming II*, MIT Press.

Langdon, W.B. and Poli, R. (1998), 'Fitness causes bloat: Mutation', in *Proc. of the 1st European Workshop on Genetic Programming*, pp. 222-230.

Langley, P., Simon, H., Bradsaw, G.L. and Zytkow, J.M. (1996), *Scientific Discovery*, New York: MIT Press.

Lones, M.A. and Tyrrell, A.M. (2002),'Crossover and bloat in the functionality model of genetic programming', in *Proc. 2002 Congress on Evolutionary Computation*, pp. 986-991.

Mecklinger, A. and Pfeifer, E. (1996), 'Event-related potentials reveal topographical and temporal distinct neuronal activation patterns for spatial and object working memory', *Cognitive Brain Research 4 (3*), pp. 211-224.

Mitchell, M. (1996), *An introduction to genetic algorithms*, New York MIT Press.

Muni, D.P., Pal, N.R. and Das, J. (2004),'A novel approach to design classifiers using genetic programming', *IEEE Trans. on Evolutionary Computation*, Vol. 8, No. 2, pp. 183-195.

O'Reilly, R.C. (1998), 'Six principles for biologically based computational models of cortical cognition', *Trends in Cognitive Sciences* 2, pp. 455-462.

Pattichis, C.S. and Schizas, C.N., (1996),'Genetics-based machine learning for the assessment of certain neuromuscular disorders', *IEEE Transactions on Neural Networks 7*, pp. 427-439.

Shallice, T., (1990), *From neuropsychology to mental structure*. Cambridge University Press.

Simon, H. A., (1996), *The Sciences of the artificial (3rd Ed.)*, Cambridge, MA: MIT Press.

Simon, H.A. (1977), *Models of discovery and other topics in the methods of science*, Dordrecht: Reidel.

Sonka, M., Tadikonda, S.K. and Collins, S.M., (1996), 'Knowledge-based interpretation of MR brain images', *IEEE Transactions on Medical Imaging* 15, pp.443-452.

Valdes-Perez, R.E. (199), 'Principles of human-computer collaboration for knowledge discovery in science', *Artificial Intelligence 107*, pp. 335-346.

Whigham, P.A. and Crapper, P.F. (2001), 'Modeling rainfall runoff using genetic programming', *Math. Comput. Model*. 33, pp.707-721.

Zubicaray, G. I. de, McMahon, K., Wilson, S.J. and Muthiah, S. (2001), 'Brain activity during the encoding, retention and retrieval of stimulus representations', *Learning & Memory* 8(5), pp. 243-251.