Learning to Recognise Mental Activities: Genetic Programming of Stateful Classifiers for Brain-Computer Interfacing

Alexandros Agapitos, Matthew Dyson, Simon M. Lucas, and Francisco Sepulveda
University of Essex
Department of Computing and Electronic Systems
Colchester, CO4 3SQ, United Kingdom
{aagapi,mdyson,sml,fsepulv}@essex.ac.uk

ABSTRACT

Two families (stateful and stateless) of genetically programmed classifiers were tested on a five class brain-computer interface (BCI) data set of raw EEG signals. The ability of evolved classifiers to discriminate mental tasks from each other were analysed in terms of accuracy, precision and recall. A model describing the dynamics of state usage in stateful programs is introduced. An investigation of relationships between the model attributes and associated classification results was made. The results show that both stateful and stateless programs can be successfully evolved for this task, though stateful programs start from lower fitness and take longer to evolve.

Categories and Subject Descriptors

I.2 [**ARTIFICIAL INTELLIGENCE**]: Automatic Programming

General Terms

Algorithms, Performance, Experimentation

Keywords

Brain Computer Interface, Classification on Raw Signal, Stateful Representation, Statistical Signal Primitives

1. INTRODUCTION

A brain-computer interface (BCI) represents a communication channel able to accept and interpret intention related commands from a human or animal brain, independent of peripheral nerves and muscles. System input is typically electrophysiological, with the use of surface electroencephalography (EEG) being most common. More invasive electrophysiological techniques such as electrocorticography (ECoG) or single unit recordings from within the cortex produce data at higher spatial and temporal resolution but yield limited contextual information whilst also inheriting risks associated with medical procedures required. The fundamental component of a BCI is the algorithm converting user input into an output signal practical for communication or control. It is understood that this process will involve the interaction of two adaptive controllers, a user conveying intentional commands and a BCI system able to differentiate said commands, the effectiveness of this interaction being a key factor in overall performance.

Genetic Programming [7] (GP) has already spawned numerous interesting applications including that of pattern classification. There are many issues ubiquitous in the design of classifier systems that make GP an attractive candidate for solving such problems. One of the crucial aspects of machine learning systems is that of hypothesis representation. The extremely flexible and expressive nature of programming languages to represent solutions to problems offers GP the capacity to represent classification problems with means unavailable to other techniques such as decision trees, statistical classifiers and NNs [9].

Currently, most evolved classifiers employ a functional expression-tree representation that does not allow the maintenance and manipulation of state information. Stateful representations have the advantage of, and potential for, integration of information over time, thus allowing for introspective rather than purely reactive classifiers. A literature review on the evolution of stateful classifiers revealed some notable exceptions on the use of memory elements within such programs. Teller's work [13] on the evolution of pattern recognisers is essentially the first attempt to include state information in the classifier programs' representation. A linear structure of scalar memory cells, named *indexed memory*, along with `read` and `write` primitive constructs was the sole machinery used to allow programs to store and retrieve state information during their computations. An additional example of state usage comes from the domain of sound discrimination using linear genetic programming [5]. The system operated an unprocessed sound signal using a single-sample moving window and allowed programs to produce an output vector of values which were fed back in at every iteration. Loveard et al. [9] employed a variant of stateful programs based on a data structure termed *certainty vector* which was used in a unilateral fashion, allowing the storage but not retrieval of information during the program's computation. This vector contained one element

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.
Copyright 2008 ACM978-1-60558-130-9/08/07...$5.00.

for each class within the problem. Once program execution halts the certainty vector is inspected and the element with the highest value is declared to be the most certain class label.

The highly stochastic nature of GP, combined with the complexity of the space of computer programs have hindered the modeling of the dynamics of GP systems; it was not until recently that light has been shed on the theoretical underpinnings of the GP approach to automatic program induction. However, this significant effort has not taken into account expression-tree representations that allow side-affecting primitives. It has been previously shown [1] that the use of state variables in evolvable programs structure the search space differently and that this can increase the evolvability of solutions within such space. It still holds that few researchers allow their GP's to include memory and there are a limited number of empirical studies that reason about either the way memory is used within evolvable individuals (see [2] for a discussion) or underlying relationships between use of memory and program fitness.

The primary motivation of this work is to extend the line of research on stateful program representations. Attempts are made to model the use of memory, hoping that this will lead to a deeper understanding of its dynamics throughout the evolutionary run and reveal relationships (if any) with program performance. As a secondary we investigate the potential of GP for both statistical feature extraction and classification based on raw EEG time-series data, strengthening the view of GP as a domain-independent approach to the signal-to-symbol problem. Two different families of classifier representations, stateful and stateless, were tested to determine whether either showed improved accuracy, precision and recall in the BCI domain. This problem domain has so far received limited attention from the evolutionary computation community as reported in [6, 3, 10]. In [6], GP was used to project EEG data into a new vectorial space of lower dimensionality to be linearly separated by a perceptron while in [10] a genetic algorithm was used for feature and parameter selection for the control of a mouse pointer. The genetic programming of an EEG classifier has only been tackled in [3] who considered a binary classification problem.

2. STATE USE DYNAMICS MODELING

In this section we are identifying a set of properties that characterise the use of memory during a program's fitness evaluation. We are aiming at developing a more clear understanding of memory usage in GP and at gaining additional insight into memory's inner workings. In order to assess the level of information provided by the various properties we calculate their correlation with classification accuracy, precision and recall.

Determining the way programs are utilizing the available memory is a difficult, open-ended process. As search spaces become larger and more complex, evolved programs become opaque to human understanding. The most important factor involves the phenomenon of *bloat*. Bloat is often facilitated by the *lazy evaluation* style of conditional (i.e `If-Then-Else`) primitives. Updates and queries of memory within the expression-tree structures are usually not easy to trace and reason about. The above factors are mainly attributed to the fact that GP does not create programs systematically but evolutionarily. It is not yet clear whether one should expect to identify common patterns of state variable usage between fragments of evolved and human-written code. Langdon [8], for example, observed unexpected results on the use of memory in evolved solutions of a FIFO list data structure. Humans often rely on *programming idioms* and conventions to generate code efficiently and reliably. Whether these are also identified and exploited through the evolutionary pathway is yet to be examined.

An interesting technique for evaluating the use of memory is reported in [12]. An individual was selected and in each set of runs a random memory index either returned a random number or zero independent of the true memory value. It was found that when all of memory indices were subjected to a random or constant damage, the individual's fitness dropped by around 10%.

In [2], Agapitos et al. surveyed the main ways that memory can be used during programs' fitness evaluation process. A *mono-phasic* fitness evaluation process considers just a single program execution per training case. It makes use of *locally scoped* variables which serve as a mean of storing the values of sub-computations at lower levels of the expression-tree which may be used multiple times as the evaluation proceeds to the upper levels. Those variables' scope is restricted to a particular expression-sub-tree. On the other hand, a *multi-phasic* fitness evaluation process considers multiple program executions per training case [1], making use of *globally scoped* variables that store the results of intermediate computations. Often, the evaluation of expression-trees that modify such global stores precedes the evaluation of those that query them. Their global visibility allows for state to be preserved between evaluations simulating the life-cycle of an object that is born at the beginning of fitness evaluation, operated upon in terms of state changes and finally dying at cessation.

The definition of the following attributes that form a model of memory usage assumes a multi-phasic fitness evaluation process and is based on the notion of *distance* between memory states formed in-between fitness evaluations. We are also assuming the use of indexed memory as the mechanism of storing intermediate information. In the context of indexed memory let *memory state* be an arbitrary snapshot of the values of the indexed elements. Let $M = \{a_i, a_{i+1}, \ldots, a_n\}$ and $M' = \{b_i, b_{i+1}, \ldots, b_n\}$ be two consecutive memory states (of n elements) at evaluation iteration t and $t+1$ respectively. The *Access Weighted Distance* ($AWD_{M,M'}$) between memory states M and M' is defined as:

$$AWD_{M,M'} = \sum_{i=1}^{N} AWD_i \tag{1}$$

where N are the available memory elements in the linear indexed memory structure and AWD_i is an augmented hamming distance between element at M_i and M'_i, defined as follows:

$$AWD_i = \begin{cases} 1 + ((WAC_i + RAC_i) \cdot weight) & \text{if } M_i \neq M'_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $WriteAccessCount$ (WAC) and $ReadAccessCount$ (RAC) are the number of times a `write` and `read` primitive respectively accesses the memory element at index i during evaluation iteration $t+1$, and M_i, M'_i are the memory elements at index i respectively. *weight* is used to discount the effect of the sum of memory access counts. Finally, the *Mean Access Weighted Distance* (MAWD) is defined as:

$$MAWD = \frac{1}{P \cdot N} \sum_{i=1}^{P} AWD_{M,M'}(i) \tag{3}$$

where P is all possible consecutive pairs of memory states during successive iterations of the multi-phasic fitness evaluation process and N are the available elements of indexed memory. MAWD captures the part of the dynamics of memory usage that deals with the amount of new information (state changes) presented to the program in every iteration, the degree to which memory registers hold input-dependent intermediate results and the degree in which a program updates and inspects the memory pool. This kind of distance is sensitive to and will detect cases where memory registers do not store any values or store constant values that do not reflect dynamic aspects (such as input based) of the computation. Intuitively, high MAWD indicates that the program is making substantial use of its memory. Based on MAWD we define the *Variance of Access Weighted Distance* (VAWD) as the average squared deviation of distance $AWD_{M,M'}$ of each pair of consecutive memory states from MAWD.

$$VAWD = \frac{1}{P} \sum_{i=1}^{P} (AWD_{M,M'}(i) - MAWD)^2 \tag{4}$$

where P is the number of consecutive pairs of memory states during successive iterations of fitness evaluations. This attribute captures the trends of fluctuation of systematicity in state changes along successive program executions. Low variance could indicate a monotony in the use of memory elements. No major variation takes place in the number of `write/read` accesses and any changes that are performed in the contents of memory registers are repeatedly observed throughout the fitness evaluation life-cycle. On the contrary, high variance indicates great variation in the number of memory accesses and a form of non-systematicity or inconsistency in the way changes to memory element values are being realized. This may in turn indicate the formation of different execution paths between subsequent fitness evaluations which may be attributed to conditional primitives leading to the evaluation of different `read` and/or `write` expression-tree nodes. Alternatively, it may signal a general difficulty of discovering a way of a consumer-producer type of memory management through a circular inspect-compute-and-update process.

An additional attribute that is part of our model of state usage dynamics is termed *Position Weighted Distance*, defined as follows:

$$PWD_{M,M'} = \sum_{i=1}^{N} PWD_i \tag{5}$$

where N are the available memory elements in the linear indexed memory structure and PWD_i is a type of hamming distance between element at M_i and M'_i, defined as follows:

$$PWD_i = \begin{cases} 1 + \frac{i - length(memory)}{length(memory)} & \text{if } M_i \neq M'_i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $length(memory)$ is the number of elements in the indexed memory structure. Note that this distance is normalized within the interval $[0,1]$ and indicates the position, within the indexed arrangement, of the memory elements that are subjected to changes. Specifically, the distance is small when differences in memory values are performed in the first indices (0,1,2,. . .) and becomes larger as state changes are being realized in subsequent ones. Based on PWD_i we define *Mean Position Weighted Distance* (MPWD) and *Variance of Position Weighted Distance* (VPWD) as follows:

$$MPWD = \frac{1}{P \cdot N} \sum_{i=1}^{P} PWD_{M,M'}(i) \tag{7}$$

$$VPWD = \frac{1}{P} \sum_{i=1}^{P} (PWD_{M,M'}(i) - MPWD)^2 \tag{8}$$

where P is all possible consecutive pairs of memory states during successive iterations of the fitness evaluation process and N are the available elements of indexed memory. What essentially interests us in the present study is VPWD so this is the sole purpose of the definition of MPWD. However, it can be argued that when combined with VPWD, MPWD itself can uncover certain properties of the program representation and population content dynamics (in terms of primitives present in the population). As already stated, when VPWD is low, MPWD can detect certain memory indices that are used in memory manipulation. These indices are the result of the evaluation of argument subtrees of `read` and `write` expression-tree nodes. The evaluation result of these subtrees is restricted by the current population content so the value of MPWD provides a form of indication of numeric value intervals that can be reached in a specific part of the fitness landscape. A quite stable MPWD over all individuals of a population along several successive generations may indicate convergence on a specific part of the search space that does not allow for the exploitation of the entire range of available memory indices. Whether this is an advantage or disadvantage is problem specific and warrants further study. For our purposes here, VPWD will be used as a modeling attribute because at the individual level it characterizes the disorder of the physical addressing of memory elements. Low variance could indicate a uniformity of the indexed positions of memory slots that are being accessed in each iteration of the fitness evaluation process. High variance indicates that no particular set of indices is being updated and accessed in each iteration and may be interpreted as a general difficulty in reading from the same memory element that information was written to. Additionally, it may signal a deficiency in determining memory areas with particular semantics so that they are consulted and updated consistently in every evaluation iteration.

The last attribute of our model is a simple measurement of the mean `read` and `write` memory accesses in each iteration. *Mean Memory Access* (MMA) is defined as follows:

$$MMA = \frac{1}{I \cdot N} \sum_{k=1}^{I} \sum_{i=1}^{N} (WAC_i + RAC_i) \tag{9}$$

where I is the number of fitness evaluation iterations, N is the available memory elements in the linear indexed memory structure, and WAC_i, RAC_i is the number of times information has been written to or read from the i^{th} memory element respectively. Note that RAC_i will be zero in the case where no information has been stored in the i^{th} element during the current or previous iterations, counteracting the undesired behaviour of the storage being read before being written. Having defined a model of the dynamics of memory usage during a program execution we will attempt to support it with empirical evidence and evaluate the importance of its attributes by investigating their relationship with program performance.

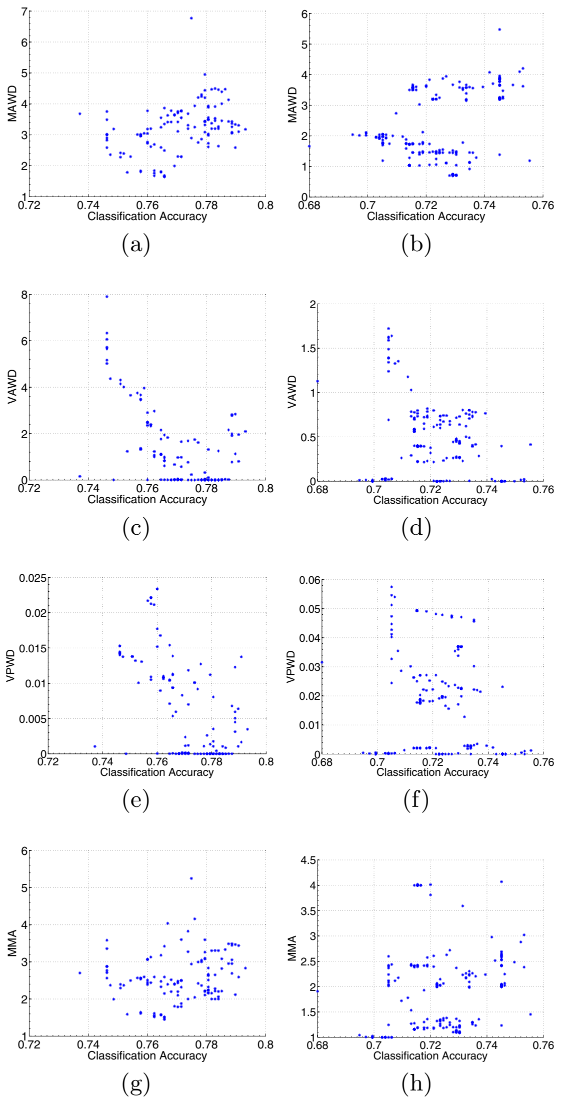


Figure 1: First row: Scatter plots between accuracy and MAWD; Second row: Scatter plots between accuracy and VAWD; Third row: Scatter plots between accuracy VPWD; Fourth row: Scatter plots between classification accuracy and MMA; (subjects 3,4)

3. METHODS

3.1 Fitness Function using Gaussian models

A novel approach for translating the numerical output of the GP classifier into a class label was introduced in [11]. A gaussian model of each program output distribution for a particular class can be acquired by evaluating the program on the example training set and calculating the mean and standard deviation of the program outputs. Assuming a binary problem case, the following equation is used to determine the *distribution distance* between classes i and j, as in [11].

$$d = 2 \times \frac{|\mu_i - \mu_j|}{\sigma_i + \sigma_j} \qquad (10)$$

where μ_i, σ_i and μ_j, σ_j are the mean and standard deviation of the program outputs for classes i and j in the training set respectively, and σ_i, σ_j are non-zero. Under this measure, for programs that distinguish between two classes well then distance d will be large, whereas the worst case is 0 where μ_i and μ_j are the same.

In multiclass pattern classification the fitness function is determined by considering the distribution distance between every two classes. For N-class problem there are $\binom{N}{2} = C_N^2 = \frac{N!}{2!(N-2)!}$ class combinations and the fitness function takes the following form:

$$fitness = \frac{1}{T}\sum_{i=1}^{T}\sum_{j=1}^{C_N^2}\frac{1}{1+d_j} \qquad (11)$$

where T is the number of training examples, N is the number of classes and d_j is the distribution distance for the class combination j.

3.2 Probabilistic pattern classification

To measure which class a given pattern belongs to, we used *multiple best programs* similarly to [11]. Assuming M best programs in the population are used the probability $Prob_c$ of a given pattern being of class c is calculated by:

$$Prob_c = \prod_{i=1}^{M} P(\mu_{i,c}, \sigma_{i,c}, o_i) \qquad (12)$$

where P is the normal probability density function, o_i is the output of program i with the pattern to be classified, $\mu_{i,c}$ and $\sigma_{i,c}$ are the mean and standard deviation (non-zero) of the outputs of program i for class c.

$$P(\mu, \sigma, o) = \frac{\exp\left(\frac{-(o-m)^2}{2\sigma^2}\right)}{\sigma\sqrt{2\pi}} \qquad (13)$$

The class with the *highest probability* is designated as the class of the pattern.

3.3 Protocol

Seven male subjects (age 24 to 35), free from medication and disorders of the central nervous system, took part in the study. Subjects were seated in an armchair with a monitor approximately 1.5 meters to their front. Within each run subjects performed a mental task in combination with an idle task. A trial began with the presentation of a fixation cross in the centre of the screen. Paired beep sounds were used to cue the active task condition and the idle task, the initial beep sound was presented at 750ms with a second at 1000ms. The first sound was consistent 1kHz, 70ms and the second alternated between 1kHz, 70ms for the active task condition and 1.3kHz, 70ms for the idle task. Subjects were instructed to attempt to perform each task until the fixation cross disappeared from the screen, occurring after 10 seconds. An inter trial period of 3 to 4 seconds was used, subjects were encouraged to use this period for blinking and to remain fixated on the cross as much as possible during trials. Each run contained ten trials (randomly ordered), trials within each run were split equally between the active task condition and the idle condition. Within each run cues appeared after a 60 second pre-trial period, the maximum length of each run was under 3.5 minutes. Each run was repeated six times, producing 30 trials for each mental task and 30 trials of associated idle data. Breaks were required every 20 minutes and granted whenever requested. All procedures were performed according to the Universities ethical regulations.

3.4 Mental activities

Left and Right Hand Motor Imagery: A wrist extension was demonstrated to the subject. Subjects were instructed to use arm-rests and perform the movement whilst concentrating on the muscular feelings associated with the action. Subjects practiced imaginary movements to their own satisfaction. It was explained to participants that visualisation of the movement was neither necessary nor likely to be beneficial.

Auditory Imagery: Subjects were required to recall a familiar tune that they knew well and "listen" to it. Instructions were given not to mouth words or make movements during the trial.

Mental Arithmetic: Subtraction was selected as the calculation task to perform. Subjects were instructed to select a three digit number and a single digit number for each trial. The single digit number was successively subtracted from the result of each calculation for the duration of the trial.

Idle Task: The idle task was undefined, subjects were instructed that during this time they should remain focussed on the fixation cross in the same manner as during an active mental task and refrain from performing any of the defined mental tasks.

3.5 Recordings

EEG was recorded from 64 electrodes positioned according to the international 10-20 layout using a BioSemi Active2 system. EEG signals were filtered between 0.1Hz and 100Hz (Butterworth - Order 5), a 50Hz notch filter was applied, data was sampled at a frequency of 256Hz. Trials were inspected for electrooculogram (EOG) artifacts at the end of session and additional runs were included if necessary to ensure data sets met a minimum size. A right ear reference was used.

3.6 Training/Test data

Data was presented to the classifiers in the form of time series of 512 signal samples. Forty-five channels were selected covering the frontal, central and temporal and parietal channels. Eight seconds of data was used from each trial (second 2 to end of trial). Of each data type 20 files were used for training, 10 for test. Data for the idle task was selected with equal proportion from active mental task sessions. In order to represent reasonable time frames for classification data from each trial was separated into seven two-second samples with one second overlap. Overlap was used to ensure the stateful classifier was not inhibited by artificial break points in the data. N-fold cross validation was not feasible due to excessive GP processing times.

3.7 Program Representation Language

Evolvable individuals employ an expression-tree representation. The primitive language is depicted in Table 3. Primitives for statistical feature extraction accept 3 arguments, the first being the time-series and the remaining two define the left and right bounds of the fragment of the time-series to consider in the statistical function. If the start and stop positions specify a negative fragment then the opposite interpretation is taken, also, if they are out of time-series length bounds, their values are induced by taking the modulus to the time-series length. The statistical function `diff` is the difference between the average values of two halves of a time-series fragment:

$$Diff(L,s,e) = \left(\frac{1}{\frac{e-s}{2}} \sum_{k=s}^{\frac{e-s}{2}+s} L_k \right) - \left(\frac{1}{\frac{e-s}{2}} \sum_{k=\frac{e-s}{2}+s}^{e} L_k \right) \tag{14}$$

where L_k is the k^{th} input list element and s, e are the start and end indices respectively. First order moment 1 `FOPDM` and second order moment `SOPDM` are *position dependent* statistics that measure how the high-valued signal samples are distributed away from the center of a time-series fragment. The first order moment is the average of the time-series values weighted by their absolute distance from the middle point of the fragment and the second order moment is the variance of these values:

$$FOPDM(L,s,e) = \frac{1}{e-s} \sum_{k=s}^{e} \left[L_k \cdot \left| k - \left(\frac{e-s}{2} + s \right) \right| \right] \tag{15}$$

$$SOPDM(L,s,e) = \frac{1}{e-s} \sum_{k=s}^{e} \left\{ \left[L_k \cdot \left| k - \left(\frac{e-s}{2} + s \right) \right| \right] - FOPDM \right\}^2 \tag{16}$$

where L_k is the k^{th} input list element and s, e are the start and end indices respectively. Finally, `read` and `write` follow Teller's definition.

3.8 Evolutionary Algorithm, Variation operators, and Run Parameters

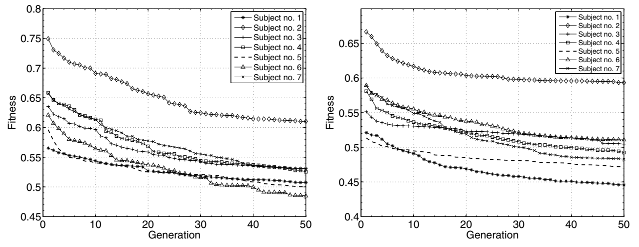
For evolutionary algorithm we used a panmictic, generational genetic algorithm combined with elitism (0.5%). The algorithm uses tournament selection with a tournament size of 4. The evolutionary run proceeds for 50 generations and the population size is set to 2, 500 individuals. Evolution

Table 1: Pearson Correlation Coefficient between classification performance and memory-usage model attributes.

Subject no. 1	Accuracy	Precision	Recall
MAWD	.6210	.5469	.6210
VAWD	−.2592	−.3086	−.2592
VPWD	.2629	.1824	.2629
MMA	.3018	.0948	.3018
Subject no. 2	Accuracy	Precision	Recall
MAWD	.1334	.1165	.1334
VAWD	.4075	.3089	.4075
VPWD	−.2426	−.1847	−.2426
MMA	.0362	.0347	.3062
Subject no. 3	Accuracy	Precision	Recall
MAWD	.3855	.2443	.3855
VAWD	−.6119	−.7307	−.6119
VPWD	−.6012	−.5539	−.6012
MMA	.1799	.1113	.1799
Subject no. 4	Accuracy	Precision	Recall
MAWD	.3802	.2796	.3802
VAWD	−.4009	−.2031	−.4009
VPWD	−.2976	−.3013	−.2976
MMA	.1393	.1770	.1393
Subject no. 5	Accuracy	Precision	Recall
MAWD	.0539	.0932	.0539
VAWD	−.0836	−.1915	−.0836
VPWD	.0249	−.0114	.0249
MMA	.0579	.0652	.0579
Subject no. 6	Accuracy	Precision	Recall
MAWD	.3994	.4682	.3994
VAWD	−.0301	−.1485	−.0301
VPWD	−.0381	.0142	−.0381
MMA	.3616	.4352	.3616
Subject no. 7	Accuracy	Precision	Recall
MAWD	.3749	.3669	.3749
VAWD	−.3185	−.4497	−.3185
VPWD	−.0227	−.0109	−.0227
MMA	−.0722	−.1514	−.0722

Table 2: Pearson Correlation Coefficient between (i) accuracy and time-series coverage percentage, (ii) accuracy and statistical feature usage

Progs.	**Accuracy**			
	Stateful		**Stateless**	
	Time-Series coverage (%)	Feature usage (%)	Time-Series coverage (%)	Feature usage (%)
1	.2212	−.0496	.3186	.4674
3	.3623	.2308	.3081	.4434
5	.4330	.3251	.3013	.4264
10	.4651	.3859	.2959	.4092
15	.4527	.3763	.2951	.4266
20	.4327	.4036	.2904	.4252
25	.3638	.4243	.2044	.3402
30	.4357	.4768	.2098	.3368
35	.3649	.4721	.2009	.3558
40	.4102	.4984	.1551	.3569
45	.3922	.4866	.1139	.3582
50	.4221	.5216	.0888	.3454
80	.3852	.5920	−.0512	.3655
100	.4223	.6048	−.3828	.3543
150	.4078	.5300	−.5869	.2923



(a) (b)

Figure 2: Learning curves for (a) stateful and (b) stateless representations

halts when all of 50 generations have elapsed. Ramped-half-and-half tree creation with a maximum depth of 7 is used to perform a random sampling of program space during the initial generation. During the run, expression-trees are allowed to grow up to depth of 17. Our search employs a mixture of mutation-based variation operators [4]: (i) **All Nodes Mutation**, (ii) **Macro Mutation**, (iii) **Point Mutation**, (iv) **Swap Mutation**, (v) **Grow Mutation:**, (vi) **Truncation Mutation**, (vii) **Gaussian Mutation**. These variation operators are applied in the following way: a sample S from a Poisson random variable with a mean of 2 was generated. S random mutation operators were uniformly picked (with replacement) from the set of available operators and were applied in sequence using a *pipe-and-filter* pattern (i.e. `Mutant=(Swap(Grow(Parent)))`). Let this type of mutation be called *Variation-Bundle*. In order to account for the exploration-exploitation trade-off we allow for the selection of either a *Variation-Bundle* or a single point-mutation (each node is being mutated with a probability of 15%) using an adaptive probability that is induced as follows: $Prob_{single-mut} = k * (gen_{current}/gen_{max})$, where $gen_{current}$ and gen_{max} are the current and maximum number of generations in the evolutionary run respectively, and k is a discount coefficient which is set to 0.6. Once the main variation procedure is performed, the mutant is subjected to a perturbation (with a probability of 50%) of the constant integer values representing the left and right bounds of the fragment of the time-series to consider in the statistical function. The rounded gaussian sample added to the constants is of zero mean and std. deviation of 12.0.

3.9 Experimental Context

Empirical results are based on the average of 10 independent runs for each subject, under each representation. For stateful classifiers, `read` and `write` primitives are included in the primitive alphabet. In addition each program has access to a linear repository of state information (indexed memory) of size 20. The evaluation of each fitness case for stateful classifiers is performed in an iterative way in order to allow for the use of intermediate state information. The recorded signal (512 samples) from each electrode is being partitioned in chunks of 50 samples with no overlap. The process initiates by initialising all memory elements to zero. In each iteration data is being fed in a moving window fashion covering each consecutive 50-sample window, allowing the program to inspect and modify previously accumulated information. The way fitness evaluation is crafted

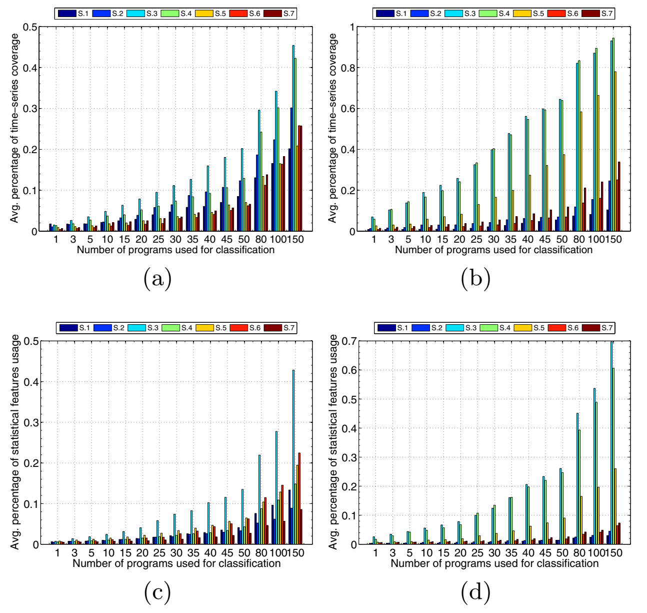


Figure 3: First row: Avg. percentage of time-series coverage in each bundle of programs used for classification; Second row: Avg. percentage of statistical feature usage in each bundle of programs used for classification; (left: stateful; right: stateless)

Table 3: Primitive Elements for Evolving Classifiers

Method	Argument(s)	Return
+, −, ∗, /, write	double, double	double
exp, ln, sqrt, sin, cos, read	double	double
mean, std.dev., skewness kurtosis, min, max, diff FOPDM, SOPDM	List, int, int	double
Terminal	**Value**	**Type**
Constant	20 rnd. in [-1,1] 100 rnd. in [-100,100]	double
	$0, \ldots, 512$	int
Parameter	45 time series	List

strongly encourages the use of memory. Programs that do not maintain state information will be only accessing the last 50 samples of the whole signal and their performance would be rather poor. The testing phase has been appropriately instrumented in order to monitor the use of memory and evaluate the significance and appropriateness of the state dynamics model introduced in section 2. For stateless representation, we followed a standard GP practice by feeding the whole 512-sample signal from each electrode during the evaluation of a single fitness case. Note that while their length varies, the number of parameter time-series was 45 in both representations.

4. RESULTS

We begin by presenting a comparison of the overall classification performance between stateful and stateless representations. Unfortunately, a table demonstrating the numerical results could not be included due to space limitations. The box-plots depicted in figure 4 show that both representations achieved competent performance and in most cases differences are of minor magnitude. A significant difference was found between accuracy and precision rates for the right hand motor imagery task between the stateful and stateless classifiers ($P < 0.05$, paired t-test, degrees of freedom $df = 6$), the stateless representation outperforming the stateful. Accuracy rates for stateful right hand motor imagery task were also significantly lower than rates for the stateful idle task ($P < 0.05$, paired t-test, $df = 6$). No significant differences were found between overall accuracy rates for the two classification representations across classes.

Across subjects the mean precision rates for the stateless classifier were found to be significantly higher ($P < 0.01$, paired t-test, $df = 34$). The only significant difference between precision rates for classes existed in the stateless condition between the calculation and idle task ($P < 0.05$, paired t-test $df = 6$). In both classifier conditions the mean precision rates, across subjects, for the calculation and auditory imagery task appear better than the motor imagery tasks, although these results are not significant. This difference in precision rates may reflect the manner in which the task is maintained over the trial period, or possibly subject

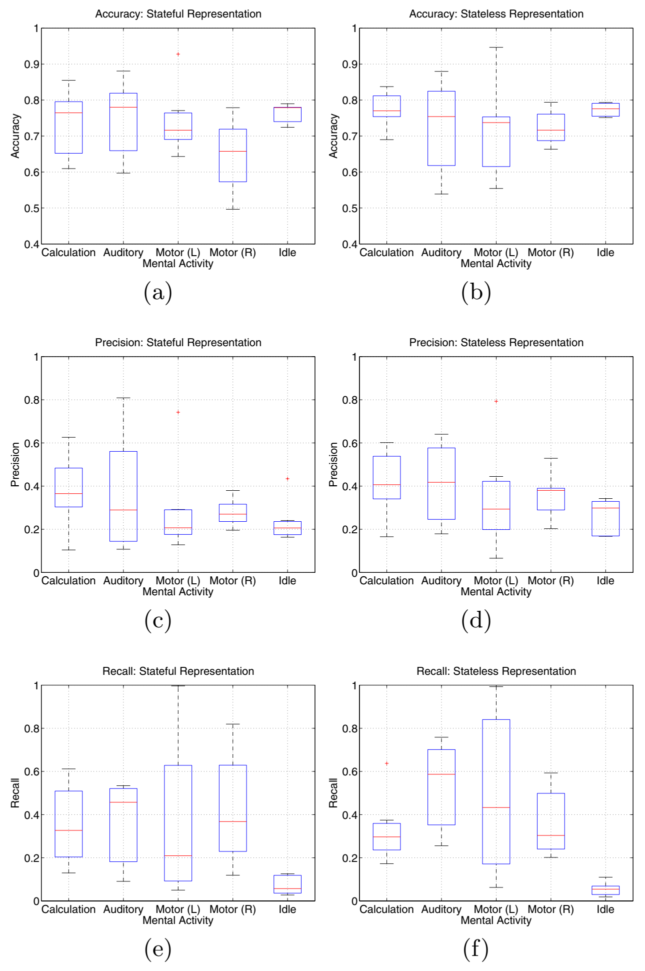


Figure 4: Box-plots: accuracy, precision, recall

familiarity with the activity. The motor imagery activities may have been executed repeatedly with specific subcomponents of the task producing variations in EEG patterns at differing times per trial, in comparison the auditory and calculation tasks may be viewed as more fluid and familiar.

Mean recall is lower for the idle state in both classifier conditions, the difference is significant in the stateful condition ($P < 0.01$, single factor ANOVA, $df = 4$) and close to significant in the stateless condition ($P = 0.055$, single factor ANOVA, $df = 4$). This result is expected as the idle task is undefined and may be representative of a number of mental activities therefore lacking the consistency found in other tasks. The stateful classifier has a higher recall rate than the stateless classifier for the auditory imagery task ($P < 0.05$, paired t-test, $df = 6$).

Figure 2 shows the average best fitness of each generation. There seem to be no severe stagnation of evolutionary improvement as evidenced by the continuous decrease of the adjusted distribution distance (representing the fitness criterion). Also, on average over the 7 subjects, stateful programs are more unfit in the beginning of the run (avg. fitness of 0.64) as opposed to stateless ones (avg. fitness of 0.57). This is intuitive, and we generally expected that initial random programs will be hardly making any sensible use of their memory. However, within the same amount of generations stateful representations (avg. final fitness of 0.52) seem to be able to compensate and reach an analogous fitness level along with stateless representations (avg. final fitness of 0.50).

We subsequently looked for correlations (Table 1) between the memory-use model attributes and classification performance. A positive correlation which is consistent across all subjects is observed between *Mean Access Weighted Distance* and classification accuracy. As one might expect, high classification accuracy is seen with programs that make substantial use of their memory. A negative correlation, which is not consistently strong across all subjects, is observed between *Variance of Access Weighted Distance* and accuracy and also between *Variance of Position Weighted Distance* and accuracy. This suggests that low variance is seen with high accuracy. Based on the discussion of section 2 this can be interpreted as meaning that efficient memory usage is associated with a uniformity of the indexed positions of memory slots that are being accessed during program execution and a consistency on the amount of read/write accesses. *Mean Memory Access* shows a weak correlation with accuracy. Interesting relations could easily exist but not necessarily be linear. The Pearson correlation coefficient used in this study only describes linear relationships so we also examine a series of scatter plots, depicted in figure 1, which can show linear relationships in addition to others. The plots are representative of a clear trend, occurring across most subjects, of high classification accuracy occurring with high MAWD/MMA and low VAWD/VPWD.

Figure 3 illustrates that the average usage of statistical features and time-series sample coverage was lower in the case of the stateful classifier. Table 2 demonstrates that the stateful representation shows higher correlation between time-series coverage and classification accuracy. In all cases of correlation (including the analysis of memory-use model) we do not infer causation, however, we intend to test the impact of selection pressure towards individuals exhibiting greater coverage of series samples and statistical features.

5. CONCLUSIONS

Genetic programming has been empirically shown to be a competent paradigm in multi-class pattern classification of raw EEG signals. Initial attempts at modeling the dynamics of state-use during program execution have been insightful in uncovering and quantifying various aspects of stateful program evolutionary induction.

6. ACKNOWLEDGEMENTS

This work was supported by the UK EPSRC under Grant EP-D03055-1.

7. REFERENCES

[1] A. Agapitos and S. M. Lucas. Evolving a statistics class using object oriented evolutionary programming. In *Proc. of 10th European Conference on Genetic Programming*, volume 4445, pages 291–300, 2007.

[2] A. Agapitos, J. Togelius, and S. M. Lucas. Multiobjective techniques for the use of state in genetic programming applied to simulated car racing. In *Proc. of IEEE CEC*, pages 1562–1569, 2007.

[3] E. Alfaro-Cid, A. Esparcia-Alcázar, and K. Sharman. A first attempt at constructing genetic programming expressions for EEG classification. In *European Symposium on Artificial Neural Networks*, pages 59–66, 2006.

[4] K. Chellapilla. Evolving computer programs without subtree crossover. *IEEE Trans. Evolutionary Computation*, 1(3):209–216, 1997.

[5] M. Conrads, P. Nordin, and W. Banzhaf. Speech sound discrimination with genetic programming. In *Proc. of First European Workshop on Genetic Programming*, volume 1391, pages 113–129, 1998.

[6] C. Estébanez, J. M. Valls, R. Aler, and I. M. Galván. A first attempt at constructing genetic programming expressions for EEG classification. In *15th International Conference of Artificial Neural Networks: Biological Inspirations*, volume 3696, pages 665–670, 2005.

[7] J. Koza. *Genetic Programming: on the programming of computers by means of natural selection.* MIT Press, Cambridge, MA, (1992).

[8] W. B. Langdon. Evolving data structures with genetic programming. In L. J. Eshelman, editor, *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, pages 295–302. Morgan Kaufmann, 1995.

[9] T. Loveard and V. Ciesielski. Representing classification problems in genetic programming. In *Proc. of IEEE CEC*, volume 2, pages 1070–1077, 2001.

[10] R. Poli, C. Cinel, L. Citi, and F. Sepulveda. Evolutionary brain computer interfaces. In *Proc. of EvoWorkshops*, volume 4448, pages 301–310, 2007.

[11] W. Smart and M. Zhang. Probability based genetic programming for multiclass object classification. In *8th Pacific Rim International Conference on Artificial Intelligence*, volume 3157, pages 251–261, 2004.

[12] A. Teller. Learning mental models. In *Proceedings of the Fifth Workshop on Neural Networks*, 1993.

[13] A. Teller and M. Veloso. Program evolution for data mining. *The International Journal of Expert Systems*, 8(3):216–236, 1995.