Dynamics of the evolution of learning algorithms by selection

Juan Pablo Neirotti and Nestor Caticha
Departamento de Física Geral, Instituto de Física, Universidade de São Paulo, Rua do Matão Travessa R 187, CEP 05508-900 São Paulo, Brazil

(Received 29 May 2002; revised manuscript received 11 December 2002; published 28 April 2003)

We study the evolution of artificial learning systems by means of selection. Genetic programming is used to generate populations of programs that implement algorithms used by neural network classifiers to learn a rule in a supervised learning scenario. In contrast to concentrating on final results, which would be the natural aim while designing good learning algorithms, we study the evolution process. Phenotypic and genotypic entropies, which describe the distribution of fitness and of symbols, respectively, are used to monitor the dynamics. We identify significant functional structures responsible for the improvements in the learning process. In particular, some combinations of variables and operators are useful in assessing performance in rule extraction and can thus implement annealing of the learning schedule. We also find combinations that can signal surprise, measured on a single example, by the difference between predicted and correct classification. When such favorable structures appear, they are disseminated on very short time scales throughout the population. Due to such abruptness they can be thought of as dynamical transitions. But foremost, we find a strict temporal order of such discoveries. Structures that measure performance are never useful before those for measuring surprise. Invasions of the population by such structures in the reverse order were never observed. Asymptotically, the generalization ability approaches Bayesian results.

DOI: 10.1103/PhysRevE.67.041912 PACS number(s): 87.23.Kg, 05.10.−a, 84.35.+i

I. INTRODUCTION

In this paper, we consider the dynamics of automatic design of learning algorithms for neural network classifiers. We use genetic programming (GP) [1] as a tool to generate a sequence of populations of programs which implement a learning algorithm. Programs at one generation give rise through crossover and mutations to offspring programs in the next generation according to their fitness. The fitness, which defines the problem to be solved, is related in this study to the efficiency of the learning algorithm implemented by the program. We choose a measure of efficiency based on the ability of generalization, related to the expected error of the output, on examples which are statistically independent from the training set. Although GP is similar to genetic algorithm (GA) [2], for both draw inspiration from Darwinian ideas, they differ in very important ways, most importantly in the representation of the evolving structures. In brief, GA aims at optimizing a fitness function defined on a parameter space of given dimension. GA deals with a population of parameter vectors in that space, which evolve by operations that typically include mutation and crossover. The optimization is carried on by selecting the fittest individuals (those vectors which give the best outcomes) for the crossover. GP, on the other hand, optimizes a fitness function acting over hierarchical structures (i.e., programs) that have no *a priori* determined form or size. These are in general represented by strings of symbols with no predefined order (which is the case for the parameter vector of GA), constrained only by syntactic rules. In GP there is a greater freedom in the structures that can be represented, whereas in GA the representation is settled from the start.

Usually the automatic design of programs has as principal aim the solution of a given problem defined by the fitness function. We are not just interested in final results, but rather in the road towards that goal and its characterization. By analyzing the dynamics we expect to learn something about how different functional structures become useful and how they invade the population of programs. In our analysis we have dealt with the following set of general questions: (a) What are the characteristics of good algorithms? or more specifically, which variable combinations are present in successful learning algorithms? (b) Is the emergence of such structures ordered in time, and if it is so, is that order robust?

We find that the road from primitive to fitter or more evolved programs is signaled by the abrupt emergence of such structures. The main result of this paper is that from our simulations we can identify such a type of time ordering. This has some rather interesting biological interpretations.

An important question deals with the choice of the learning scenario. We study learning of a static linearly separable (LS) rule, a sufficiently simple learning problem that can be studied by analytical means, as far as final results are concerned, but which presents a wealth of interesting results. Extensions to non-LS and time dependent rules, the evolution of kernels for support vector machines and several other possible extensions are left for future work. Related questions have been addressed before [3,4], see about best results and Bayesian bounds in Ref. [5], for a variational point of view about the perceptron learning in Ref. [6], about feedforward architectures with hidden units in Refs. [7–9], for drifting rules in Refs. [10,11], in an unsupervised scenario [12], from a more general Bayesian perspective in Refs. [13–15]; in the case of offline learning in Refs. [5,16]. From the perspective of time ordering it has been discussed in Ref. [17].

Analysis of the emergence of structures and ensuing invasions is done by characterization of the populations at two different levels: phenotypic and genotypic. At the phenotypic or expressed level, description deals with quantities that

measure the expression of important traits. Program differences are irrelevant as long as they give rise to the implementations of the same function. Even different functions are the same if they lead to similar fitness values. The genotypic level deals with properties that depend on the detailed structure of the individuals. Two programs are genotypically different if their sequences of variables and operators are different, even if they give rise to the same outcomes. Two entropies are used in order to deal with the different description levels. The phenotypic entropy S describes the distribution of fitness in the population, and the genotypic entropy H describes the distribution of probabilities of symbols [18].

The creation of a new program by mixing or crossover of preexisting programs, although not impossible, can be difficult to implement in programming languages such as C and Fortran. The main problem is that the program that manipulates the population programs—the *metaprogram*—has to cut and paste the parsing trees of the population programs, and then compile them in order to perform the fitness measure, all during run time [1,19]. The major part of the work in GP has been developed in LISP which is also the case in this study. LISP's most prominent characteristic with regard to GP is that programs and data have a common form and are treated in the same manner. This common form is equivalent to the parse tree for the computer program. Thus, it can be found that genetic manipulations of parse trees are natural LISP operations. We have developed also a protocol for simulation of LISP on a Linux cluster, which is described in Ref. [20].

The paper is organized as follows. In Sec. II A, we describe the learning scenario of rule extraction by a percpetron learning from examples. Section II B gives a brief description of GP from the very special point of view which interests us here. Section II C follows with a description of the tools to characterize the dynamics. Section III presents the results and concluding remarks can be found in Sec. IV.

II. THE PROBLEM AND THE METHOD

A. Problem: Learning by a perceptron

The learning problem to be analyzed by the GP must strike a balance between being complex enough so that interesting dynamics arises and being simple to the point so that details can be understood and simulations performed. The perceptron meets these demands and has a long and distinguished history. For an extensive view from a statistical mechanics perspective, see Ref. [4]. We consider the realizable teacher-student learning scenario. The perceptron classifies vectors $\mathbf{S}\in\mathbb{R}^N$ (here obtained i.i.d from a uniform distribution) in two categories with labels $\sigma_{\mathbf{J}}=\pm 1$ according to the rule $\sigma_{\mathbf{J}}=\mathrm{sgn}(\mathbf{J}\cdot\mathbf{S})$. The aim of any learning dynamics is to determine the weight or synaptic vector $\mathbf{J}\in\mathbb{R}^N$ from pairs of examples $(\mathbf{S}_\mu,\sigma_{\mathbf{B}\mu})$ which carry information about a rule. We restrict ourselves to the case of noiseless realizable rules, with the labels uncorrupted and generated by another perceptron with an unknown weight vector $\mathbf{B}\in\mathbb{R}^N$. We consider on-line learning, so $\mathbf{J}$ will be built sequentially by modifications induced by the arrival of new pairs of examples. This can be simplified even further by concentrating on the particular form of Hebbian learning, where the increments of $\mathbf{J}$ are weighted by a *modulation* function f, thus

$$\Delta\mathbf{J}_\mu=\mathbf{J}_\mu-\mathbf{J}_{\mu-1}=f\sigma_{\mathbf{B}\mu}\mathbf{S}_\mu/\sqrt{N}. \tag{1}$$

This is not very restrictive, as a large fraction of the previously studied algorithms, both on-line and off-line, may be put in a similar way (see, e.g., Ref. [4]) and in the (thermodynamic) limit of large networks it can represent asymptotically efficient learning, which even saturate Bayesian bounds.

There are two time scales in this problem. The slow scale is measured in generations, where the learning programs evolve. The fast scale can be thought of as the *age* of a neural network. It is a measure of the number of examples to which a network has been exposed in the learning phase, where fitness is being measured.

B. Method: Algorithm construction by genetic programming

In this section, we describe briefly our implementation of GP for the problem at hand. Conventional GA works by manipulating fixed-length character strings that represent candidate solutions of a given problem. For many problems, hierarchical computer programs are the most natural representation for the solution. Since the size and the shape of the program that represents the solution are unknown in advance, the program should have the potential of changing its size and shape.

The simulation starts with a population of randomly created programs. All these programs have been constructed using predetermined sets of variables and operators. The construction process respects some rules in order to avoid the creation of programs that cannot be evaluated. The GP operations are used to create the population of the next generation. The programs are ranked by their fitness and then the GP operations are applied again. These two steps are then iterated.

The most common computer language used in GP is LISP, therefore we will refer to the population individuals as programs or LISP S expressions indistinctly. We call a *faithful S expression* (FSE) a list of symbols that do not return an error message when evaluated. Components, also called atoms, of the S expressions can be either functional operators or variables. Let $\mathcal{F}$ be the set of all the operators and $\mathcal{V}$ the set of all the variables used to write down the FSEs. The choice of these sets depends upon the nature of the problem being faced. For instance, if the solution of a problem can be represented by quotients of polynomials, a suitable choice for the sets is $\mathcal{F}=\{+\ -\ */\}$ and $\mathcal{V}=\{\mathrm{x1}\}$. For example, a FSE is $(+(+\mathrm{xx})\ (*\mathrm{x}(-\mathrm{x}(-\mathrm{xx}))))$, which is a (nonunique) LISP representation of the function $2x+x^2$. The simplest FSE is a variable. The next simplest FSE is an operator followed by the appropriate number of variables (two in the example above). All FSEs are either a variable or a list with an operator followed by an appropriate number of FSEs. Unfaithful S expressions are for instance (xx), $(+\mathrm{x}*)$, and $(\mathrm{x}-\mathrm{x})$.

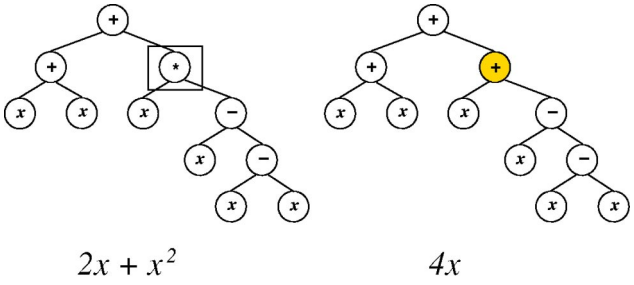

FIG. 1. LISP programs as parsing trees before and after a GP mutation. A randomly selected atom in the parse tree is changed to another randomly selected atom of the same type. In this example, a multiplication operation is replaced by an addition.

The GP operations considered in the present work are asexual reproduction, mutation, and crossover. During asexual reproduction a certain fraction of the top ranked individuals, the top set, is copied without any modification into the new generation, ensuring the preservation of structures that made them successful. Mutation is implemented on an individual by changing an atom at a random position. The new and old atoms are different but of the same kind to ensure faithfulness. Finally, the modified tree is copied into the new generation. In order to accelerate the dynamics, different mutation rates can be used for different atom types.

There are no sexes associated to the programs and crossover is a hermaphrodite sexual GP operation. Crossover parents are chosen by tournament, which is done as follows. First, consider a set of the population, then select an age a such that $1 \leqslant a \leqslant P$ (P is the maximum age) with a probability proportional to a. From the chosen set, a certain number (e.g., ten) individuals are selected at random. The program with smaller generalization error at age a is selected for crossover. In our experiments, the first parent is chosen from the top set by tournament [1]. The second parent is chosen by tournament among the entire population. From each parent, an atom of the same type is selected at random. The FSEs with roots in the selected atoms are interchanged to generate two offsprings. In order to avoid uncontrolled growth if the depth of any of the offsprings is above a given threshold, the program is deleted. The mutation and crossover operations are represented in Figs. 1 and 2.

The GP parameters used in the simulation are shown in Table I. A short discussion about the parameter values will be presented in the conclusions. At generation zero, a population of 500 FSEs is created at random. The programs have (in agreement with Table I) a maximum depth of seven nested parentheses. The variable set used to build the programs is

$$\mathcal{V}=\{\sigma_{\mathbf{J}\mu}\sigma_{\mathbf{B}\mu}h\mathbf{S}_\mu\mathbf{J}_\mu\}, \tag{2}$$

the presumed and correct classifications, the postsynaptic field

$$h=\mathbf{S}_\mu\cdot\mathbf{J}_\mu/\|\mathbf{J}_\mu\|, \tag{3}$$

the input and the synaptic vectors. The operator set is

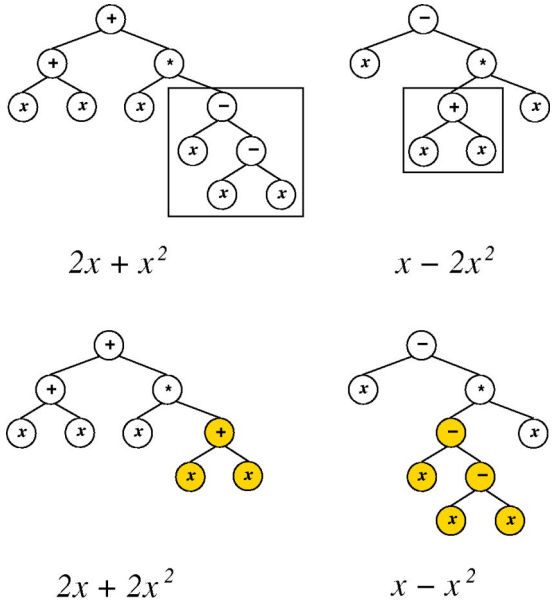

FIG. 2. GP crossover. Two parents are selected from the population. A random point in each tree is selected. The branches that grow from the point are interchanged in order to generate two offspring.

$$\mathcal{F}=\{\mathsf{Psqr}\ \mathsf{Pexp}\ \mathsf{Plog}\ \mathsf{abs}\ +\ -\ *\ \%\ \mathsf{p.}\ \ \mathsf{pN.}\ \ \mathsf{ev}*\ \mathsf{vv}+\ \mathsf{vv}-\}, \tag{4}$$

where Psqr, Pexp, Plog, and % are the protected square root, exponential, logarithm, and division; abs, $+$, $-$, and $*$ are the usual absolute value, addition, subtraction, and multiplication; and p., pN., ev*, vv+, and vv− are the inner product (e.g., $\mathbf{x}\cdot\mathbf{y}$ for $\mathbf{x}$, $\mathbf{y}\in\mathbb{R}^N$), normalized inner product ($\mathbf{x}\cdot\mathbf{y}/N$), the product of a scalar times a vector ($a\mathbf{x}$), the addition of two vectors and the subtraction of two vectors ($\mathbf{x}\pm\mathbf{y}$), respectively. Protected functions are functions whose definition domains have been extended in order to accept a larger set of arguments. The definitions of these functions appear in Table II.

TABLE I. Control parameters for the GP simulation in our experiments.

Parameters	Values
Population size	500
Reproduction rate	10%
Mutation rate	0.01%
JJ mutation rate	0.2%
Max. depth generation-0	7
Max. depth generation-G	17
Probability of internal point selection (crossover)	90%
Tournament participants	10
Vector sizes	11
Maximum number of training examples	100
Maximum number of sets of examples	50
Slave processors	10

TABLE II. Definition of the protected function as FSEs. The protected square root is just the square root of the absolute value of its argument. In this manner we extended its domain into the negatives. The exponential is well defined in the reals, although, in order to avoid overflows, we have to impose a cutoff. The protected logarithm has a cutoff at a small positive number to extend its domain to the nonpositive numbers. And the protected quotient allows the division by zero (if the absolute value of the denominator is smaller than a tiny number, the protected quotient returns a big number, if not it just returns the usual quotient).

Function	Definition
(Psqr x)	(sqrt (abs x))
(Pexp x)	(exp(min13.0 x))
(Plog x)	(log(max1.d−17 x)
(%x y)	(if(>1.d17(abs y))1.d17(/x y))

If either one of the offsprings has a depth bigger than 17, it is deleted. With a mutation rate of 0.01% (one every 20 generations) a mutation is performed in the offsprings. Because the pairs $\mathbf{J}_\mu\mathbf{J}_\mu$ become rare after few generations (at the beginning of the simulation, the learning algorithms that use $\mathbf{J}_\mu$ are not efficient) we keep injecting this pair with a rate of 0.2% (at least one individual per generation receives this pair). Different mutation rates just serve the purpose of accelerating the dynamics and decrease the time scale, which it takes for interesting things to happen. The process is repeated until the new population reaches the full size fixed here at 500.

After a new population is created, the fitness of each individual is measured and so a new ranking is built. There is a great freedom in choosing the fitness function. It is always a macroscopic or phenotypic quantity, i.e., a function of the expressed characters, and although it reflects the microstructure, it is not a function of the genetic details of the individual. Errors in the measurement of the fitness have a bearing on the dynamics, not entirely different from the temperature in simulated annealing.

C. Method: Characterization of the dynamics

We are interested in studying the evolution of learning algorithms and merit can be attributed in different ways. We choose to study the case where merit is based on the ability to generalize. The generalization error is

$$e_g(\mu)=\langle\Theta(-\sigma_{\mathbf{B}\mu}\sigma_{\mathbf{J}\mu})\rangle, \tag{5}$$

averaged over the distribution of examples. In the thermodynamic limit, we have the following for uniform distribution of examples:

$$e_g(\mu)=\frac{1}{\pi}\arccos\rho, \tag{6}$$

where

$$\rho=\lim_{N\to\infty}\left(\frac{\mathbf{J}_\mu\cdot\mathbf{B}}{\|\mathbf{J}_\mu\|\|\mathbf{B}\|}\right),$$

and μ indicates its age, e.g., how many examples have been presented to the network. Since the aim is to obtain algorithms with the smallest possible generalization error at all ages, and given that the expected generalization error (at least for good learning algorithms) is a decreasing function of the age, a suitable choice for the fitness function of the kth program is

$$F^{(k)}=\sum_{\mu=1}^{P}\mu e_g^{(k)}(\mu), \tag{7}$$

where P is the total number of examples (maximum age) presented to the network. To calculate the generalization error, an average is taken over at least 50 sets of examples.

The distribution of fitness across the population can be described by the normalized fitness, a measure of the fraction of the total (exponential) fitness that an individual has,

$$n^{(i)}=\frac{\exp(-F^{(i)})}{\sum_{k=1}^{M}\exp(-F^{(k)})}, \tag{8}$$

where $F^{(k)}$ is the fitness measure of th kth individual of the population Eq. (7). Note that smaller values of the fitness are associated to better performances. The use of the exponential just amplifies the importance of the individuals with better performance. We introduce the entropy of the normalized fitness

$$S=-\sum_{k=1}^{M}n^{(k)}\ln(n^{(k)}), \tag{9}$$

a function of the expressed characters of the population (fitness), thus dubbed the phenotypic entropy or Ph-entropy. Note that this entropy is largest when all the members of a population have the same fitness and that the appearance of a distinguished individual, for better or worse, is signaled by a decrease in the Ph-entropy.

Each FSE in the population has a well defined length $\lambda^{(k)}$, that is, the number of atoms (operators and variables) that make it up. We define the mean length L as

$$L=\frac{1}{M}\sum_{k=1}^{M}\lambda^{(k)}. \tag{10}$$

To characterize the internal structure of the programs, we estimate for each position i the probability that symbol s_q (a variable or an operator) appears at position i, $\omega(s_q|\,i)$, by measuring the frequency over all the population:

$$\omega(s_q|i)=\frac{\sum_{k=1}^{M}\Theta(\lambda^{(k)}-i)\,\delta(s_i^{(k)}|s_q)}{\sum_{k=1}^{M}\Theta(\lambda^{(k)}-i)}, \tag{11}$$

where $\delta(s_i^{(k)}\,|\,s)=1$ if $s_i^{(k)}$, the ith symbol in the kth individual, is equal to s, and zero otherwise. The genotypic en-

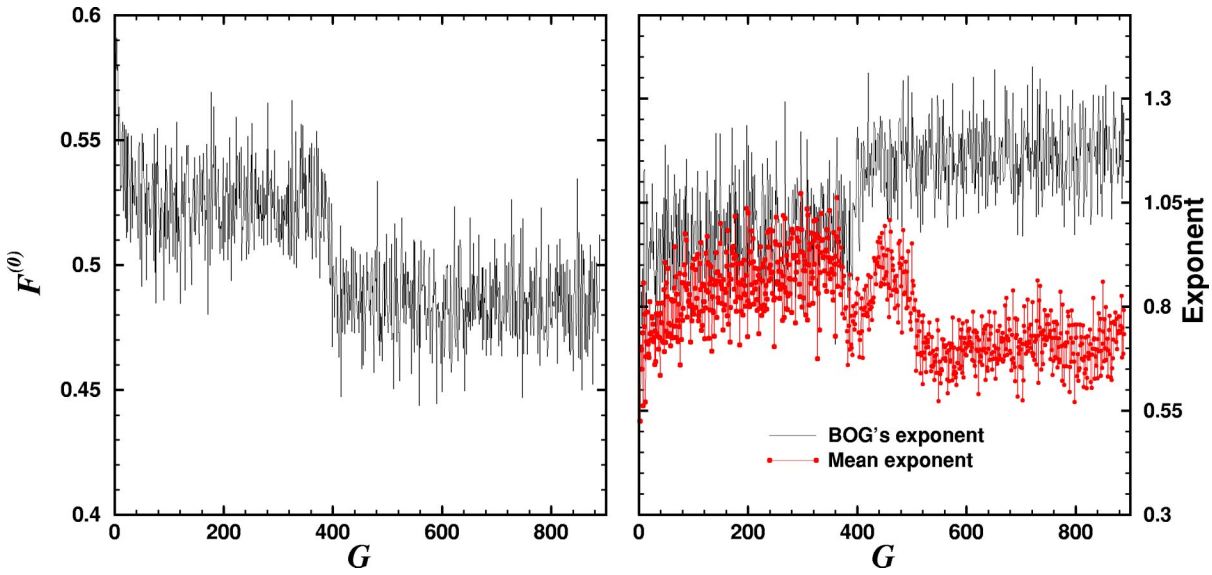

FIG. 3. (Left) Fitness of the best-of-generation individual of the population vs the number of generations. A sudden change takes place around 380 generations. (Right) The exponent of algebraic decay of e_g. Upper curves, BOG, lower curves, population average. All quantities here and in the following figures are dimensionless.

tropy (or G-entropy), which is a function of the microstructure of the individuals in the population, is then defined as [18]

$$H = -\sum_{s_q \in \mathcal{Q}} \sum_i \omega(s_q|i) \ln_{|\mathcal{Q}|} \omega(s_q|i), \tag{12}$$

where $\mathcal{Q} = \mathcal{F} \cup \mathcal{V}$ is the set of symbols.

A measure of the capacity of a population of using a functional structure may be given by the frequency where the combination of variables is found. This is admittedly crude, since its position in the program determines whether it is useful or not. On the other hand, the absence of such combination does not rule out the possibility that some other combination is doing the job in a more cumbersome manner.

There are two quantities or functional structures that are of interest both in a quantitative and a qualitative analysis of the learning algorithms. The first can be associated to the product $h\sigma_{\mathbf{B}\mu}$. This quantity can be functionally described as quantifying a measure of surprise. This is because if $h\sigma_{\mathbf{B}\mu} > 0$ ($\sigma_{\mathbf{B}\mu} = \sigma_{\mathbf{J}\mu}$), the network will classify correctly the example with classification label $\sigma_{\mathbf{B}\mu}$, while if $h\sigma_{\mathbf{B}\mu} < 0$ ($\sigma_{\mathbf{B}\mu} \neq \sigma_{\mathbf{J}\mu}$), the classification is wrong. Thus it gives a signal of how wrong or correct was the classification and also how stable that classification is under changes of the weight vector. This is obviously an important factor to take into account while incorporating the information in to a given example. Thus, we define the surprise $\overline{\mathcal{S}}$ as the following mean value:

$$\overline{\mathcal{S}} = \frac{1}{N_P} \sum_{k=1}^{M} \sum_{i=1}^{\lambda^{(k)}-1} [\delta(s_i^{(k)}|h)\delta(s_{i+1}^{(k)}|\sigma_{\mathbf{B}\mu}) + \delta(s_i^{(k)}|\sigma_{\mathbf{B}\mu})\delta(s_{i+1}^{(k)}|h)], \tag{13}$$

where N_P is the total number of pairs of symbols of the population.

The second functional structure we will concentrate on is something that can estimate the performance or acquired experience of the network in the implementation of the rule. If properly used, this is akin to annealing of the learning rate or to the functional annealing in learning algorithms. This can be implemented by using the length of the weight vector $\mathbf{J}_\mu$. Therefore, we will associate to performance $\overline{\mathcal{P}}$ the following expression:

$$\overline{\mathcal{P}} = \frac{1}{N_P} \sum_{k=1}^{M} \sum_{i=1}^{\lambda^{(k)}-1} \delta(s_i^{(k)}|\mathbf{J}_\mu)\delta(s_{i+1}^{(k)}|\mathbf{J}_\mu), \tag{14}$$

that is, the density of pairs $\mathbf{J}_\mu \mathbf{J}_\mu$.

III. RESULTS

Our numerical experiments have been performed in a Linux cluster, using the strategy described in Ref. [20]. Fifteen experiments, starting from different random seeds, have been performed using the GP paradigm described above. Each run could take up to a week. Some runs failed to evolve to anything interesting. On three runs, the programs collapsed to just one symbol. For six of them, the complexity of the populations was high but no good solutions were found and no abrupt changes in behavior occurred. Interesting results were obtained in the remaining six. Within this last group, results were different in several respects, but most interestingly, had striking similarities which we now describe. In what follows we consider an illustrative run which presents clearly some features that are typical of the other runs of the last group. We found a dramatic change of behavior around generation 380 that can be seen by using several different signatures. Figure 3, left side, shows the fitness

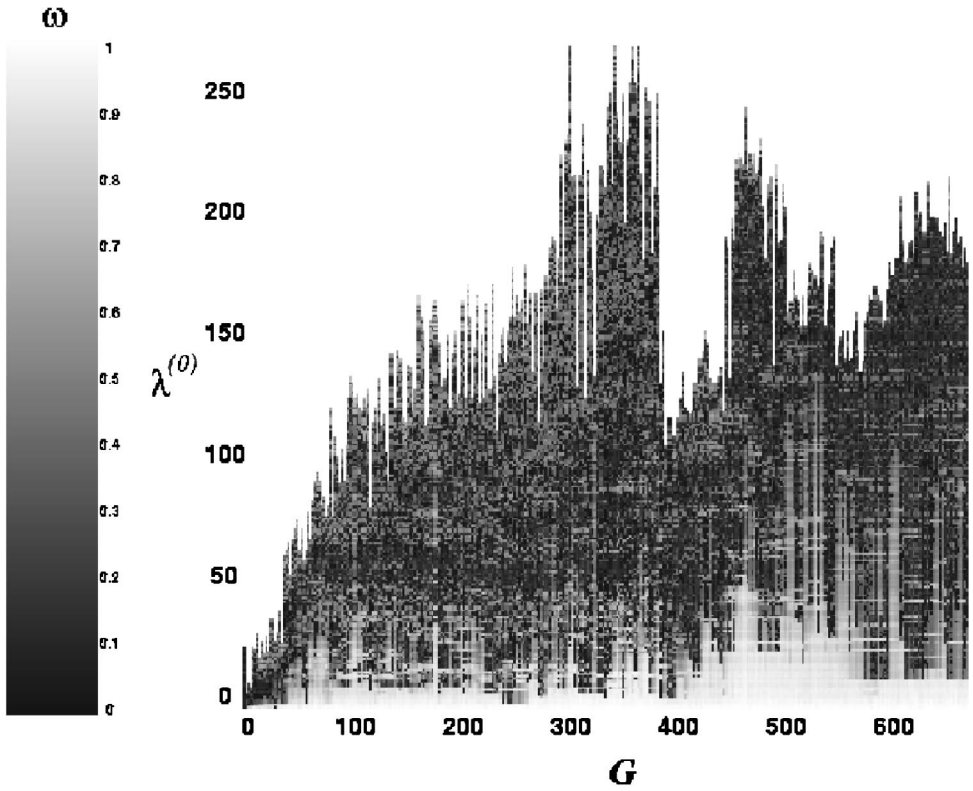

FIG. 4. Gray-scale coded bar graph of the best-in-generation (BOG) individual. The time G in the horizontal axis is measured in generations and the length of the program is in the vertical axis. Gray level of pixel at coordinates (G,i) codes for the frequency $\omega(s_q|i)$, at which the symbol s_q (which is the ith atom of the best-of-generation FSE) appears at position i, at generation G according to the scale on the left.

of the most adapted program or best of generation (BOG) as a function of time. On the right side the figure shows the exponent of the generalization error e_g decay of the BOG and of its average over the population. Any reasonably fit algorithm should present a generalization error that decays with age, for otherwise the network would not be learning and thus unfit. For randomly chosen examples we can expect on general theoretical grounds a power law decay [4]. The exponent shows a sharp change, specially if the population average is compared to that of the BOG. Finite size errors are responsible for the fact that exponents larger than 1 can be found. To understand how representative of the whole population is the BOG we composed a gray-level coded bar graph (see Fig. 4) where each vertical bar represents the BOG program written as a string of symbols; time is measured in generations in the horizontal axis. At the position of each symbol in the program, a gray square represents the empiric probability of the symbol in the population Eq. (11). Note that quite rapidly (in no more than 20 generations), an initial symbol is predominant in the population. This is invariantly found in all runs and it is always a symbol that ensures that the modulation function is positive, for otherwise the learning would be anti-Hebbian and inefficient. The initial part of the code is very robust and thus is shared by almost all the population.

Both entropies (phenotypic and genotypic) present changes about the same time (Figs. 5 and 6). The scale of the fluctuations of the Ph-entropy [Eq. (9), Fig. 5] after the transition are much larger than before. A decreasing trend in the Ph-entropy can also be identified. Low values of this entropy identify the existence of phenotypically distinguished programs. The decreasing trend after the transition together with the decrease of the average decay exponent shows that although the BOG has been largely improved, the population in general has not. The G-entropy Eq. (12) and the mean length Eq. (10) change abruptly at the transition. This reduction is not present in all transitions. It just indicates that in this particular run the invading mutation occurred in a small program. The G-entropy and the mean length are linearly

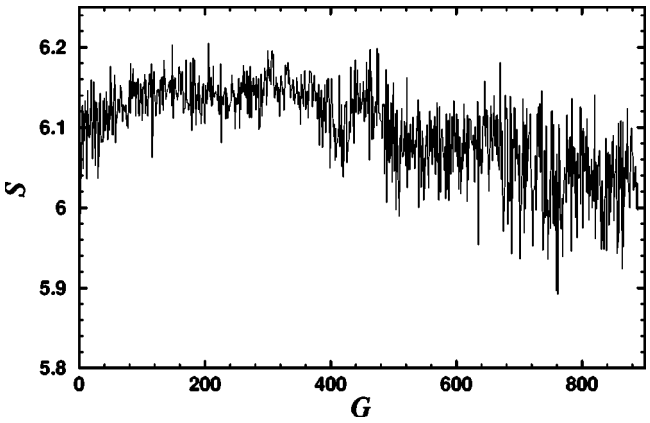

FIG. 5. Phenotypic entropy as a function of the number of generations G.

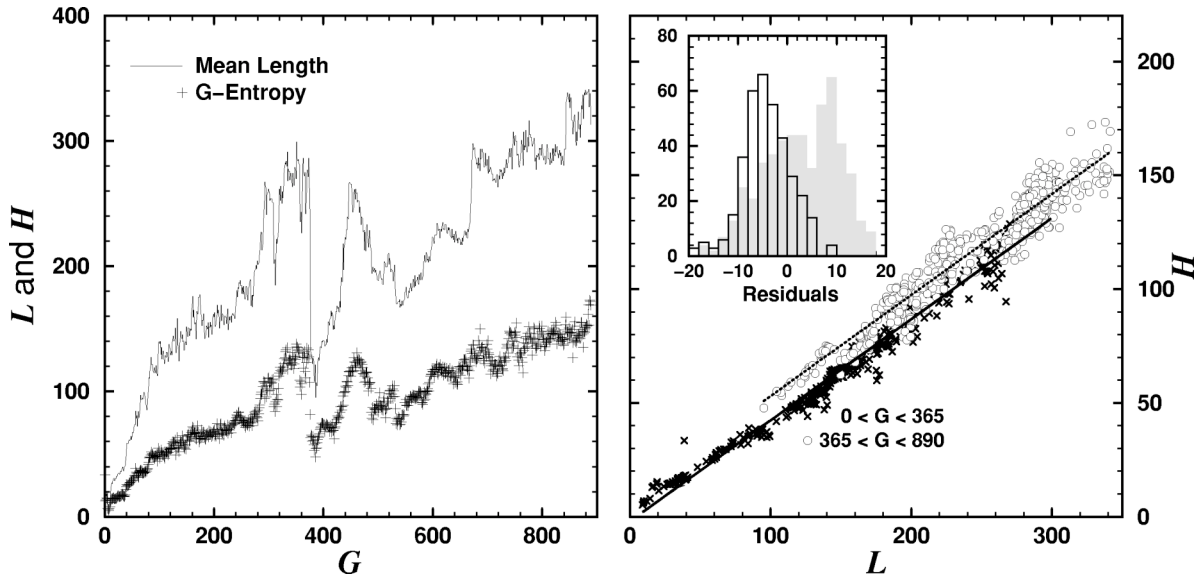

FIG. 6. (Left) Genotypic entropy and mean length as functions of the number of generations G. The transition can be seen by the sharp change around $G=380$. (Right) H vs L. Two linear fits are shown for data before the transition (crosses) and data after the transition (circles). To see that two linear fits are necessary we did a single linear fit of the whole data set and plotted (inset) the histograms of the residuals to the single linear model. The two histograms are for data before and after the transition, respectively, and the separation of the two peaks lends support to the modeling by two linear regimes.

correlated. This is natural since G-entropy, as defined should be extensive. What is unexpected is the fact that there are two distinct linear regimes before and after the transition. To see this we did a linear fit to the whole data set and plotted a histogram of the residuals, that is, the difference between the actual value of a data point and the corresponding value of the linear model. The two histograms in the inset of Fig. 6 show clearly a systematic error for the single linear model. These results prove the existence of two different regimes. If the whole simulation is considered, fluctuation distributions of both entropies apparently have long tails, but looking at each regime separately, a simple Gaussian distribution of different width also fits the data.

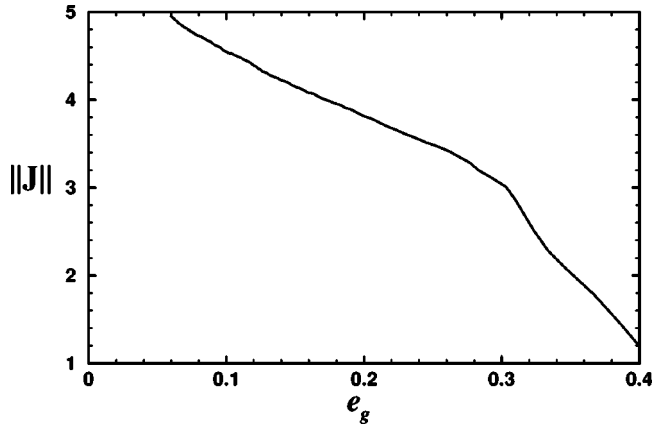

FIG. 7. Typical behavior for late stage generation: the length of the weight vector $||\mathbf{J}||$ increases monotonically when the error of generalization decreases, thus it can be used as a measure of the experience of the individual or of its performance in solving the classification problem. It leads to efficient annealing of the learning rates.

To understand the nature of the regimes mentioned above, we present Figs. 7–11. To show the correlation between the size of the synaptic vector and the generalization error for later stages of the simulation, we show a graph of $\|\mathbf{J}_\mu\|$ as a function of the generalization error for a program with a good fitness (Fig. 7). The two variables are clearly anticorrelated. This justifies our interpretation of the probability defined in Eq. (14) as a means of measuring performance. In Fig. 8 we plot the surprise $\overline{\mathcal{S}}$ Eq. (13) and the performance $\overline{\mathcal{P}}$ Eq. (14), as functions of the number of generations. It is clear that the emergence of a measure of performance is at about 380 generations.

In Fig. 9 we present the most adapted individual at generations 300, 350, 400, and 450. Just before the transition there is no pair $\mathbf{J}_\mu\mathbf{J}_\mu$ present in the program (at generations

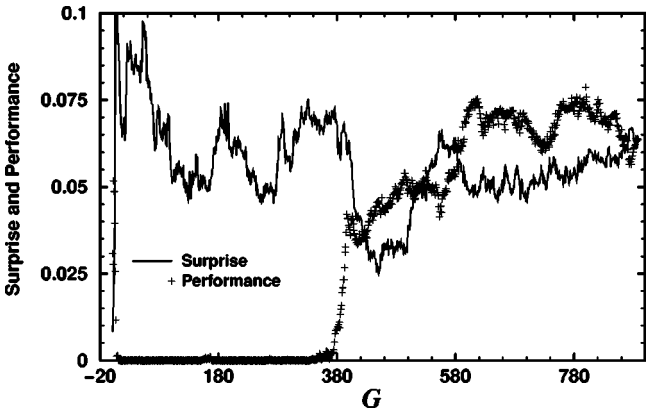

FIG. 8. Typical behavior of the density of pairs $\overline{\mathcal{S}}$ (surprise) and $\overline{\mathcal{P}}$ (performance) as functions of time G measured in generations. Notice the sharp rise at the beginning of $\overline{\mathcal{S}}$ and the later rise of $\overline{\mathcal{P}}$. The time ordering is robust and was never seen in the reverse order.

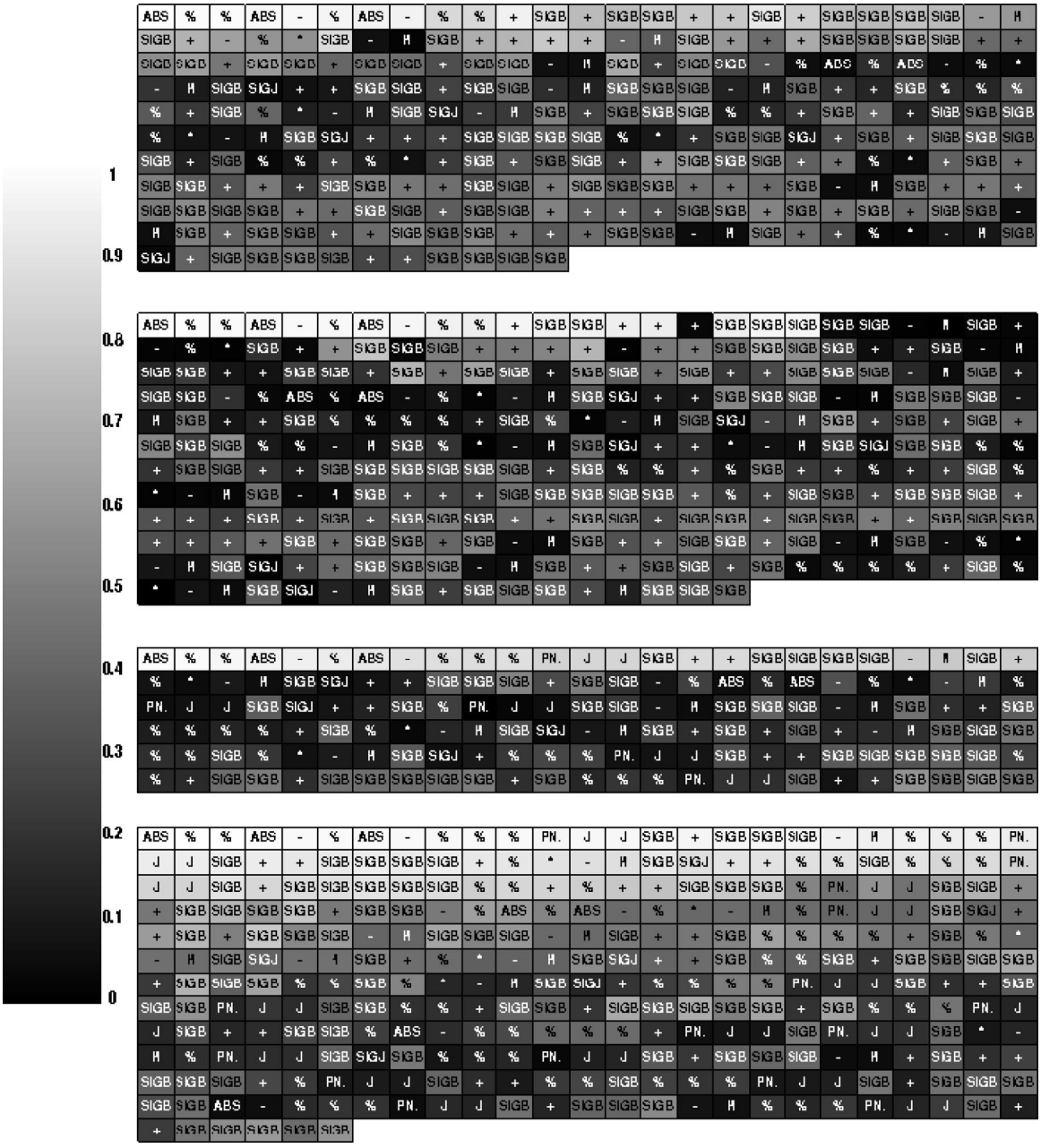

FIG. 9. The strings of symbols are the programs best of generation at generations 300, 350, 400, and 450 (from top). The gray levels represent the frequencies $\omega(s_q|i)$, according to the gray-level scale.

300 and 350). After the transition, the size of the best individual decreases and several pairs $\mathbf{J}_\mu\mathbf{J}_\mu$ appear.

A more general analysis of the density of pairs can be done with the help of Fig. 10. In these pictures we present the relative frequencies at which each possible pair appears in the population. The vertical axis represents the first element of the pair, the horizontal axis the second element. The size of the white squares represents the frequency of the pair, relative to the most frequent pair (represented by the largest square in each picture). In panel (a) we present the density of pairs at generation 300, (b) corresponds to generation 350, (c) to generation 400, and (d) to generation 450. In (a) and (b) there is no measure of performance. The most frequent pair is the combination $\sigma_{\mathbf{B}\mu}\sigma_{\mathbf{B}\mu}$, which is just a 1, but not quite since it can evolve into different directions. After the transition, in panels (c) and (d), this pair remains the most frequent, but important changes have happened. There are small white squares for the pair $\mathbf{J}_\mu\mathbf{J}_\mu$ representing the emergence of a measure of performance $\overline{\mathcal{P}}$ by the learning algorithms.

The modulation function of the best individual at the beginning of the simulation, before the transition, and after the transition, is presented in Fig. 11. The Hebbian regime, the regime with surprise, and the regime with both surprise and performance clearly appear in panels (a), (b), and (c), respectively. The modulation function in (c) closely resembles, in shape and fitness, the optimal Bayes result.

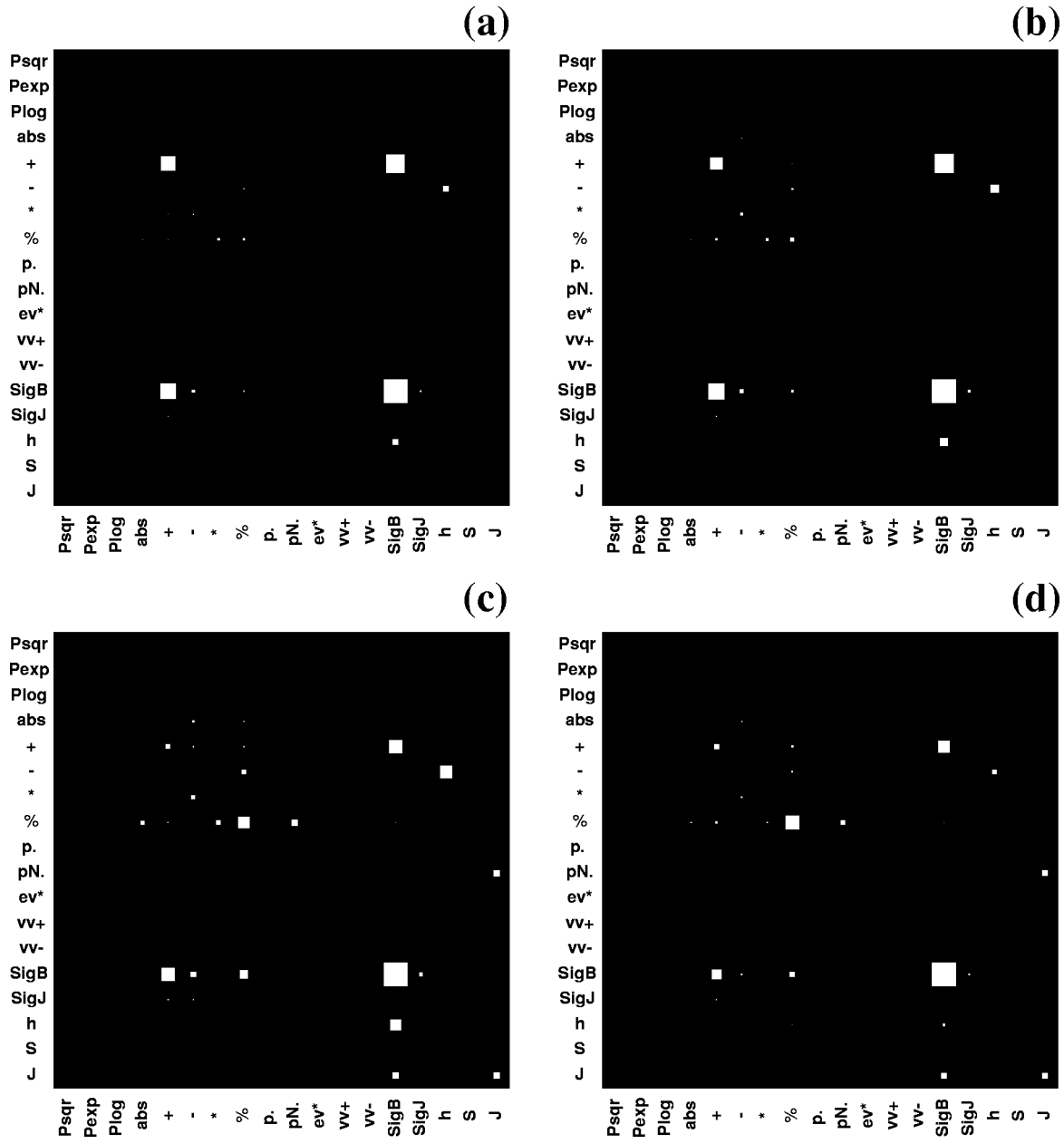

FIG. 10. Density of pairs for generations 300 (a), 350 (b), 400 (c), and 450 (d). In all the cases the most frequent pair is $\sigma_{\mathbf{B}\mu}\sigma_{\mathbf{B}\mu}$ (SigB SigB). Only in the last two panels does the pair **JJ** appear.

IV. CONCLUSIONS

Evolutionary programming techniques provide the means to automatically design programs which solve certain class of problems. In this paper, however, we were not only interested in the final result. The problem that GP was set out to solve has been previously analyzed from many angles and a detailed understanding of on-line learning in perceptrons has been achieved. Rather we concentrated on the dynamics of evolution and have detected dynamical changes in the behavior of the GP solutions that are related to the emergence of functional structures. This is not a conventional phase transition associated to singularities arising in the thermodynamic limit. Nevertheless, the abruptness of the invasions, measured in generations, justifies calling it a transition.

A few runs failed to present the transition, possibly because of time limitations, but it was seen in many different runs. Some features were never reproducible but others were present in every experiment. As examples of those features that depend upon contingencies we include the number of generations before the change takes place, the width of the change (some were just about ten generations wide, others took several tens of generations), and the result of the GP itself, i.e., the program that implements the best learning algorithm. The variability of these features, important as they may be from a constructive point of view when the solution to the problem is the main concern, indicates that taking averages over different runs would lead to wrong interpretations. Standard statistical assesment of the probability of oc-

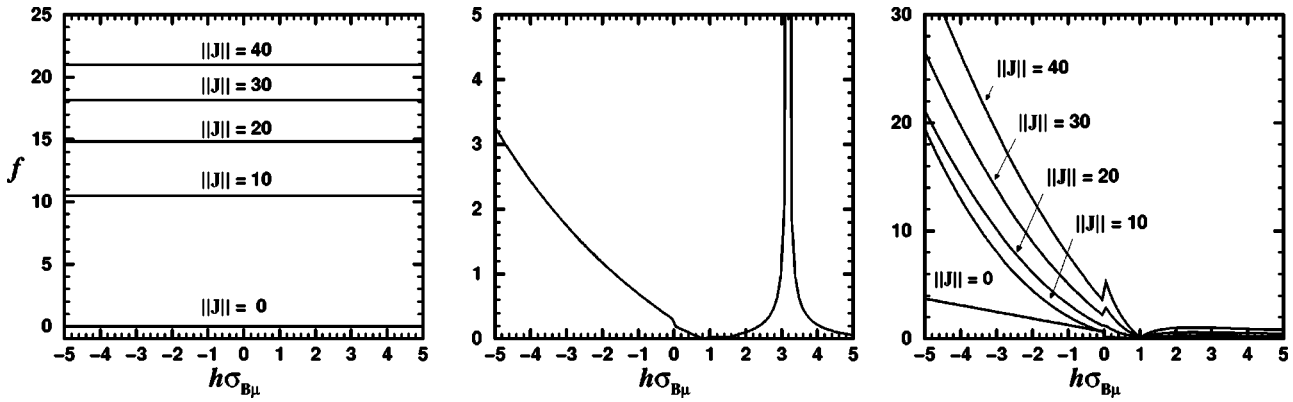

FIG. 11. Modulation functions. (Left) Early stage where surprise is not measured and annealing by experience is ineffective. (Center) Intermediate stage, now surprise is used but annealing by experience has been lost. (Right) Late stage, after the transition, where surprise through the measurement of $h\sigma_{\mathbf{B}\mu}$ and annealing $\mathbf{J}_\mu\mathbf{J}_\mu$ are correctly implemented.

currence of the phase transition is left for future work. We tried, instead, to identify robust features which can be confidently expected to occur every time good solutions were obtained. In serving such purpose we have characterized the dynamics by looking at the phenotypic Eq. (9) and genotypic entropies Eq. (12), which give a picture of the distribution of phenotypic fitness and functional or symbolic structure, respectively. The conformation diagram Fig. 4 gives a bird's eye view of the relation of the BOG and the frequency of symbols in the population as well as its length. A more complete analysis of the BOG can be done through Fig. 9. The gray scale attributes light gray to symbols extremely frequent in the population at that position, and monotonically attributes increasingly darker gray levels as the symbols get more unlikely to be found. We can see that after the change, the third program (BOG at $G=400$) presents symbols mostly in the dark end of the scale, except for the consolidated initial part. Fifty generations later, there are islands of lighter gray in the BOG. That means that the genetic character of the best individual has invaded the population. The main robust feature can be identified once the change has been understood from a functional point of view, in terms of two concepts: the surprise ($\overline{\mathcal{S}}$) that newly arrived information elicits and how such information should be taken into account based on how much experience ($\overline{\mathcal{P}}$) the network has in solving the task at hand. A temporal order in the emergence of structures can be identified. Performance can be useful only after surprise is measured correctly.

It can be shown, at least in the thermodynamic limit, that for algorithms which do not measure surprise their generalization error decays as $\mu^{-1/2}$ and for them annealing is useless. Learning algorithms that use surprise have a faster decay ($e_g \propto \mu^{-1}$) and algorithms that use both surprise and annealing by experience have the fastest decay since they can have smaller coefficients of μ^{-1}. From Figs. 8 and 11 we can conclude that the chronological order is respected. At earlier stages, the BOG is unable to use surprise. Although surprise functional structures are found throughout the population, their incorrect use makes the BOG an annealed Hebbian algorithm. It is known that annealing will not improve the Hebbian learning and the frequency of performance functional structures decreases until it vanishes. Thus $\overline{\mathcal{P}}$ is almost immediately extinguished from the population. It will only appear in very modest ways through mutation and, repeatedly individuals which use it become extinct. Later on, surprise is finally well accounted for and correctly classified examples cause typically smaller Hebbian corrections than those incorrectly classified. At that point the correct use of surprise potentializes the beneficial use of functional structures that measure performance. Then a correctly annealed algorithm emerges that resembles quite closely the modulation functions found through Bayesian or variational approaches. This successful strategy invades the population.

The values of the parameters used to perform the experiments described (Table I) are the best values found, according to Ref. [1] and previous experiments. Populations larger than 500 increase the required CPU time for the experiment, without improving the description of the phenomena. The reproduction rate, maximum tree's depths, and probability of internal point selection are the same as in Ref. [1], which presents a reasonable justification for these values. The value of the mutation rate has been set low enough in order to avoid deleterious effects. The rate of the specific **JJ** mutation is appropriate in order to have a mutant per generation. This accelerates the emergence of the performance and does not affect the time ordering. The number of tournament participants has been increased from a low value of 2 to 10 in order to study the invasion of the best individual genes into the population (observe that the bigger the number of participant the larger the over selection of the best individuals). The effects observed are not strongly coupled with the number of tournament participants, thus we left this number set to ten. The size of the perceptron, the number of training sets, and the number of examples per set were adjusted in order to get smooth generalization error curves.

There are several possible extensions of this problem. From a biological point of view, there is a suggestive similarity with the time order in which certain structures responsible for measuring surprise and performance have appeared. Will this order be found in more complex artificial settings? Is this biologically significant? Can it be extended to other functional structures? It should also be quite interesting to

further analyze other transitions in the automatic design of programs.

ACKNOWLEDGMENTS

The simulations described here were done on a cluster made possible through the efforts of J. L. deLyra, C. E. I. Carneiro and co-workers. The cluster's construction was partially supported by FAPESP and CNPq. J.P.N. received financial support from FAPESP and N.C. received partial support from CNPq. Discussions with Osame Kinouchi and Mauro Copelli were important during the earlier stages of this work.

[1] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, MA, 1992).
[2] J. H. Holland, *Adaptations in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence* (University of Michigan Press, Ann Arbor, 1975).
[3] S. Amari, IEEE Trans. Electron. Comput. **16**, 299 (1967).
[4] A. Engel and C. Van den Broeck, *Statistical Mechanics of Learning* (Cambridge University Press, Cambridge, 2000).
[5] M. Opper and D. Haussler, Phys. Rev. Lett. **66**, 2677 (1991).
[6] O. Kinouchi and N. Caticha, J. Phys. A **25**, 6243 (1992).
[7] M. Copelli and N. Caticha, J. Phys. A **28**, 1615 (1994).
[8] R. Vicente and N. Caticha, J. Phys. A **30**, L599 (1997).
[9] R. Simonetti and N. Caticha, J. Phys. A **29**, 6243 (1996).
[10] M. Biehl and H. Schwarze, Europhys. Lett. **20**, 733 (1992).
[11] O. Kinouchi and N. Caticha, J. Phys. A **26**, 6161 (1993).
[12] C. Van den Broeck and P. Reimann, Phys. Rev. Lett. **76**, 2188 (1996).
[13] M. Opper, Phys. Rev. Lett. **77**, 4671 (1996).
[14] M. Opper, in *On-line Learning in Neural Networks*, edited by D. Saad (Cambridge University Press, Cambridge, 1998), p. 363.
[15] S. Sara and O. Winther, in *On-line Learning in Neural Networks* (Ref. [14]), p. 379.
[16] O. Kinouchi and N. Caticha, Phys. Rev. E **54**, R54 (1996).
[17] N. Caticha and O. Kinouchi, Philos. Mag. B **77**, 1565 (1998).
[18] This is similar to that introduced in C. Adami, C. Ofria, and T. C. Collier, Proc. Natl. Acad. Sci. U.S.A. **97**, 4463 (2000), but not restricted to fixed size program length.
[19] *Advances in Genetic Programming*, edited by K. E. Kinnear, Jr. (MIT Press, Cambridge, MA, 1994).
[20] J. P. Neirotti and N. Caticha, available at http://www.fge.if.usp.br~nestor/