Computing and Informatics, Vol. 37, 2018, 1073–1102, doi: 10.4149/cai_2018_5_1073

A NEUROGENETIC ALGORITHM BASED ON RATIONAL AGENTS

Lídio Mauro Lima de Campos

Faculty of Computing/ICEN
Universidade Federal do Pará, Rua Augusto Corrêa 01, Guamá
CEP 66075-110, Caixa postal 479, Belém, Pará, Brasil
e-mail: `limadecampos@gmail.com`

Roberto Célio Limão de Oliveira

Faculty of Computer Engineering
Universidade Federal do Pará, Rua Augusto Corrêa 01, Guamá
CEP 66075-110, Caixa postal 479, Belém, Pará, Brasil
e-mail: `limao@ufpa.br`

Gustavo Augusto Lima de Campos

Faculty of Computing
Universidade Estadual do Ceará, Centro de Ciências e Tecnologia
Av. Paranjana 1700, Itaperi, CEP 60740-000, Fortaleza, Ceará, Brasil
e-mail: `gustavo@larces.uece.br`

Abstract. Lately, a lot of research has been conducted on the automatic design of artificial neural networks (ADANNs) using evolutionary algorithms, in the so-called neuro-evolutive algorithms (NEAs). Many of the presented proposals are not biologically inspired and are not able to generate modular, hierarchical and recurrent neural structures, such as those often found in living beings capable of solving intricate survival problems. Bearing in mind the idea that a nervous system's design and organization is a constructive process carried out by genetic information encoded in DNA, this paper proposes a biologically inspired NEA that evolves ANNs using these ideas as computational design techniques. In order to do this, we propose a Lindenmayer System with memory that implements the principles of organization, modularity, repetition (multiple use of the same sub-structure), hierarchy (recursive composition of sub-structures), minimizing the scalability problem of other methods. In our method, the basic neural codification is integrated to a genetic algorithm (GA) that implements the constructive approach found in the evolutionary process, making it closest to biological processes. Thus, the proposed method is a decision-making (DM) process, the fitness function of the NEA rewards economical artificial neural networks (ANNs) that are easily implemented. In other words, the penalty approach implemented through the fitness function (Equation (5)) automatically rewards the economical ANNs with stronger generalization and extrapolation capacities. Our method was initially tested on a simple, but non-trivial, XOR problem. We also submit our method to two other problems of increasing complexity: time series prediction that represents consumer price index and prediction of the effect of a new drug on breast cancer. In most cases, our NEA outperformed the other methods, delivering the most accurate classification. These superior results are attributed to the improved effectiveness and efficiency of NEA in the decision-making process. The result is an optimized neural network architecture for solving classification problems.

Keywords: Evolutionary computation, neural networks, grammatical evolution, hybrid intelligent systems

1 INTRODUCTION

The design of an Artificial Neural Network (ANN) can be considered a complex decision making process that frequently relies on the user experience. In general, this ANN design task is a trial and error process, where a number of different transfer functions and amount of hidden neurons should be adjusted in order to solve a specific problem. As the design of ANN is still being done, manually, by human experts, the automation of this design process will benefit the decision-making process done by human experts [2].

Bio-inspired algorithms have shown efficient in different nonlinear optimization problems [3, 5, 13, 44, 30]. Due to their efficiency and adaptability, the interest in research in the field of neuroevolution (i.e. evolutionary approach to design ANNs) has increased lately. The core issue in neuroevolution is to build an efficient indirect encoding scheme [6, 7, 8, 9, 10]. Which means that the evolutionary algorithm evolves a compressed description of the ANN rather than the ANN itself. [8].

These approaches are biologically motivated by the fact, that in case of the human brain, there are more neurons than nucleotides in the genome. Moreover, in biological genetic encoding the mapping between genotype and phenotype is indirect. The phenotype typically contains orders of magnitude more structural components than the genotype contains genes. Nevertheless, neuroevolution has not so far addressed that the development and evolution of neurons is carried out by genetic information encoded in DNA and when followed will generate the final shape of the organs including the brain.

ADEANN is inspired by two natural biological mechanisms: genetic encoding and the evolution of genetic coding. As is well-known, neuron development is governed by the genetic information encoded in deoxyribonucleic acid (DNA), and ultimately generates the final shape of the brain. During biological development, in the complex process that links the DNA code to its phenotype, the same gene is used in many contexts. This compactness minimizes the information required to describe complex individuals. On the other hand, evolution describes the temporal changes in the genetic code (DNA). Among the several mechanisms underlying these evolutionary changes, natural selection is very important.

The two natural processes described above are hybridized such that the DNA contained in cells can also spawn cells. On the other hand, the changes in DNA are passed onto later generations. Motivated by these natural processes, we propose an artificial hybrid system that abstracts these natural mechanisms at an acceptable level of complexity.

To this end, we propose a new NEA, a biologically inspired artificial hybrid system [45, 46, 47, 49, 50] called Artificial Development and Evolution of ANNs (ADEANN). The ADEANN integrates two components. The first is a generative representation that represents genotypes (a set of production rules of a Lindenmayer system) by a compact indirect encoding scheme (IES). The IES also conducts and controls the process of mapping the genotypes to the phenotypes (complex neural morphologies). To mimic the DNA encoding scheme and enable scalability, our IES leverages the phenotype representation to a smaller genotype. Thus, the search process is carried out in a lower-dimensional solution space. In addition, our IES implements the organizational principles of hierarchy, modularity, and gene reuse (allowing compact representation of complex phenotypes). The second component is a genetic algorithm (GA), a simplified representation of natural evolution. In local search problems based on GAs, a bit string is called a chromosome (the genotype). Each bit on the chromosome is a gene, and a gene set represents the parameters of a function to be optimized. Each string is assigned a fitness that indicates the quality of its encoded solution (the phenotype). To improve the biological realism of GA, the GA in our approach evolves the generative representation. The evolutionary process can be regarded as the temporal genetic changes in the hypothetical DNAs of a population of individuals, regulated by an artificial selection mechanism. The above biological inspiration underlies the originality of our approach. To our knowledge, we report the first attempt to generate recurrent neural networks from combined metaphors.

The main contribution of our method is the genotype representation by our proposed IES. Using a compact DNA encoding, we codify a parametric Lindenmayer system (L-system) with memory, which implements the principles of organization, modularity, repetition, and hierarchy to achieve complex neural architectures (multilayer and recurrent networks). [34, 35] adopted L-systems, although [35], used DNA encoding, their study was restricted to feedforward neural networks, whereas our approach is extended to recurrent networks. In the IES used by [34], the genotypes encode twenty rewrite rules of an L-system. Our DNA encoding system encodes a parametric L-system with memory using 10 production rules. Therefore, our IES is more compact than [34]'s method, and reduces the search space of all feasible solutions. In addition, the memory mechanism in our approach enables the reuse of phenotypic structures (rewrite of nodes and connections) at different stages of development. Such reuse is an important capability of NEAs.

Our ADEANN also utilizes expert knowledge of the problem to more efficiently search the infinite space of topologies, thus minimizing the expert's effort in the optimization. The penalty approach implemented by the fitness function (Equation (5)) automatically rewards the economical ANNs with stronger generalization and extrapolation capacities. Our L-system generates ANN topologies without requiring additional substrate configurations for the given problem.

This paper is organized as follows. Section 2 discusses the state of the art. In Section 3 we describe a new approach to formalize the problem of ADANNs (artificial development and evolution of ANNs) as a local search based on rational agents. Section 4 introduces a biologically inspired method for automatic design of ANNs. In Section 5, the experimental setup and results are presented. Lastly, a discussion and conclusions are presented in Sections 6 and 7, respectively.

2 STATE OF THE ART

This section discusses existing research on NEAs. Most NEAs use direct encoding systems (DESs) [12], which specify every connection and node in the genotype that will appear in the phenotype (ANN). These methods are simply implemented, but the size of the connectivity matrix scales as the square of the number of nodes. In NEAT [6], the DES incorporates a few biologically plausible entities, and alters both the weighting parameters and structures of the networks. [29] developed a novel multi-objective optimization for a hierarchical genetic algorithm (MOHGA) based on the micro-GA approach. However, this method was not tested in other applications such as temporal series forecasting.

As discussed by [1], the increasing complexity of evolutionary computation demands more sophisticated methods than direct mapping from genotype to phenotype. At the other extreme are indirect encoding systems (IESs) [7, 28, 27, 8, 9, 10, 16, 30]. In an IES, the network generation is indirectly specified by the genotype. The solution description is compressed, enabling complex topologies of ANNs.

ES-HyperNEAT [10, 19] has shown promising results enabling the development of complex regular plastic ANNs. The encoding scheme adopted by HyperNEAT [8, 20] and ES-HyperNEAT [10] does not shape itself to the biological development process, including the nervous system. The CPPN (Compositional pattern-producing networks) in HyperNEAT plays the role of DNA in nature, but at a much higher level of abstraction; in effect, it encodes a pattern of weights that is painted across the geometry of a network. Furthermore, in these methods both the genotype and phenotype are neural networks, which is not biologically plausible.

The ES-HyperNEAT [10] is an improvement over HyperNEAT [8]. While the location of the hidden nodes in the substrate had to be decided by the designer on the original HyperNEAT, ES-HyperNEAT showed that the decision might in fact be automated. In the HyperNEAT, CPPN encodes an infinite number of weights within the Hypercube, from which a subset should be chosen to be incorporated into the ANN. [10] consider other important information is the density of nodes from which an additional increase of the same does not offer any advantage. For more details about state of the art, query the authors' research [21].

The following methods [7, 9, 30], belong to the class of GDSs (generative and developmental systems) using grammatical evolution (GE). The methodologies took a step towards of biologically inspired approaches. Lee et al. [9] sought inspiration from the DNA, in their research, information is coded using the symbols A, G, T and C. A sequence of three of these symbols is known as a codon. The sequence of codons between these delimiters is translated into a production rule for developing a neural controller. The production rule of the L-System used by [9] is context-free and does not allow to generate recurrent networks. A similar idea [7] using binary strings also exists in which the process of reading and translating bits can be repeated from different bits in the string to produce different production rules. The ANE proposed by [7] has one disadvantage that is the difficulty to set the parameters of the fitness function of Genetic Algorithm to direct the search for minimum architectures.

The data structure of NEAT [6], which represents the genotype, grows linearly with the number of edges between two nodes. (MOHGA) [29] cannot yield the recurrent neural networks (RNNs). Despite the novel capabilities of HyperNEAT, the user must manually place the hidden nodes at the substrate, which substantially disadvantages the original HyperNEAT. The applicability of the method proposed by [30] to other applications, such as temporal series-forecasting, was not tested, and the model cannot yield RNNs.

3 OUTLINE OF THE APPROACH

Our approach involves the formulation of an artificial neural network design problem (ANNDP) as an optimization problem, that is: given a set of L observations on the behavior of a particular process, $\Psi = \{(xd^l, yd^l)\}$, $l = 1, \ldots, L$, where xd^l represents a numeric vector defined in R^n and yd^l is a numeric vector defined in R^m, the task is to find a back-propagation ANN's topology, $yc^l = \mathrm{ANN}(w*, xd^l)$, which minimizes the mean square error between yd^l and yc^l, this is, between the desired values in the observations set and the computed values in the neurons' outputs situated in the ANN's output layer.

An ANN's topology can be described as a finite set of neurons, that is, nodes in graphs notation Nodes $= \{n_1, n_2, \ldots, n_k\}$, and a finite set $H \subseteq N \times N$ of connections between neurons, meaning, directed edges in graphs notation. From the point of view of graph theory, feed-forward ANNs (FANN) are acyclic graphs while recurrent ANNs (RANN)are cyclic graphs. An input layer is a set of input units, that is, a subset of n nodes whereas an output layer is a set of output units, namely a subset of m nodes. In FANNs, the k^{th} layer ($k > 1$) is the set of all nodes $n_i \in$ Nodes. This type of nodes have an edge path of length $k-1$ between some input unit and u. In fully connected RANNs, all units have connections to all non-input units.

We approach the ANNDP solving based on the problem-solving-agent proposed by Russell and Norvig [23], whose agent is called ADEANN (artificial development and evolution of ANNs), which encapsulates a special scheme of solutions representation as well as a local search strategy based on genetic algorithms to solve the problem. Regarding the representation scheme, the approach adopts a generative representation, that is, instead of an encoded ANN topology, each chromosome stores a set of production rules of a Lindenmayer system, which, in turn, generates ANNs topologies. With regard to the solution process, the SEARCH-ANN function outlined below illustrates the structure of the program in the ADEANN agent.

Function SEARCH−ANN(SearchParameters, TransitionModel, FitnessFunction) **return** an ANN topology

```
1: inputs: SearchParameters, TransitionModel, FitnessFunction
2: vars: $Pop$, $t$, $PopPerformances$;
3:
4: $k \leftarrow 0$
5: ANNsPop$^k$ ← Generate-ANNs(SearchParameters)
6: ANNsPerformance ← Evaluate-ANNs(ANNsPop$^k$, FitnessFunction)
7: loop do
8: if StopConditionTest($k$, ANNsPerformance, SearchParameters)
9: then return solution(Best-ANN(ANNsPop$^k$, ANNsPerformance)
10: ANNsPop$^{(k+1)}$ ← (ANNsPop$^{(k+1)}$, ANNsPerformance, TransitionModel,
SearchParameters)
11: ANNsPerformance ← Evaluate-ANNs(ANNsPop$^{(k+1)}$, FitnessFunction)
12: $k \leftarrow k+1$
13: end
```

The SEARCH-ANN function starts the local search process aiming to achieve an artificial neural network topology $yc^l = \text{ANN}\{(w^*, xd^l)\}$, which minimizes the mean square error between yd^l and yc^l, for $l = 1, \dots, L$ in the ANNDP's formulation. This function employs information on the search parameters (SearchParameters input term) as well as a transition model (TransitionModel input term) to describe how to modify current populations of ANNs and generate a new population, in addition to an evaluation function (FitnessFunction input term) to measure the value of each ANN in a current population.

Firstly, in the beginning of the process, Generate-ANNs function generates an initial population of ANNs, where each ANN is represented by a set of production rules of a Lindenmayer system codified in a chromosome. This function considers the information in the SearchParameters input term on the desired number of ANNs in the populations as well as on the desired length for the chromosomes in the population. Evaluate-ANNs function stores in ANNsPerformace the computed performance value of each ANN topology in the current population based on the mean square error computed in the output layer of the ANN. SEARCH-ANN function employs an iteration counter (k) and a condition named StopConditionTest Boolean function to decide when to stop the local search process and return a solution to a problem. Stop condition is described by means of a proposition relating the information on the current iteration counter k and the information available in SearchParameters input term, that is, regarding the max number of loops in its repetition scheme, which is central to the local search strategy in the approach, as well as on an ideal performance value such that for an ANN to be considered a solution.

Modify-ANN function is executed repeatedly seeking to transform a current population of ANN's in a new population of ANNs. In our approach, this function encapsulates the evolutionary principles of pairs selection and crossing over pairs and individual mutation. Central to the approach, compact indirect encoding scheme (IES) conducts and controls the process of mapping a set of production rules of a Lindenmayer system codified in a chromosome to an associated ANN topology.

It is worth mentioning that the local search strategy in the SEARCH-ANN function is an adaptation of the generate-and-test programming technique in order to find the best ANN topology. More specifically, in this case, the strategy can be said to consisting in an adaptation of the modify-test programming technique, such that the goal is to find the best ANN, according to the values of FitnessFunction, as well as that the next solution set is the transformation of a given current solution set.

4 BIOLOGICALLY INSPIRED NEA

The general structure of ADEANN is shown in Figure 1. The optimization process of ADEANN proceeds through several stages. The GA starts with a population of individuals randomly initialized with 0 s and 1 s (Figure 1 a)). Second, the bits of each individual of the population are subjected to transcription (Figure 1 b)) and translation (Figure 1 c)), following valid production rules. Both processes are performed by the function Rule-Extraction-with-GA, presented in Section 4.2. After finding the appropriate production rules (Table 1), the rewriting system generates the genotypes (Figure 1 d)). All of the genotypes are mapped to phenotypes (ANN architectures) (Figure 1 e)). The ANNs are then trained (Figure 1 f)) and validated, and tests are carried out. The classification accuracy of each ANN is measured from its fitness (Figure 1 g)), calculated by Equation (4) or Equation (5). The latter equation implements a penalty approach that rewards economical ANNs with stronger generalization capacity and greater ease of implementation. The genotypes are classified by the performances of their ANNs (Figure 1 h)). The GA selects the best individuals (Figure 1 i)) for mutation and crossover operations (Figure 1 j)), which provide the new population (Figure 1 k)). The previous steps are repeated through n generations.

4.1 Neural Structure Codification with L-System

To mimic the mechanism of grown structures, including neurons, we adopt a parametric L-system with memory. It comprises a set of rules created from an alphabet. This system can be described as a grammar $G = \{\Sigma, \Pi, \alpha\}$, where the alphabet consists of the elements of the set $\Sigma = \{., f, F, n, [,], *, B\}$, and the production rules (Π) described in Table 1. The axiom $\alpha = .$ is the starting point of the developmental process, where f denotes a neuron and F is a connection between neurons, [and] indicate storage and recovery, respectively, of the current state of the development, $*$ denotes that the string is recovered from storage, and B is the connection of a neuron with a block of neurons. The second rule $. \rightarrow (f \ldots f)n$ means replace the start point by the neurons of the input layer. Rule 3.1 ($f \rightarrow [f$) means to store the position of the current neuron, so as to start a new ramification from it. Rule 3.2 ($f \rightarrow fFf$) means establish a connection between two neurons. Rule 3.3 ($f \rightarrow fF$) means establishing a connection from a specific neuron. Rule 3.4 ($f \rightarrow n$) means replace a provisional neuron with a permanent neuron. Rule 3.5 ($f \rightarrow f$) means to maintain a specific neuron during development. Rule 3.6 ($f \rightarrow fB$) means connect a neuron to a block of neurons. Rule 4 ($[\rightarrow [Ff]$) means start the development of a new ramification from a specific neuron and recover the previous state. Rule 5 ($f \rightarrow f*$) means recover a previous ramification stored for use.

Our approach is biologically motivated by the human brain, which contains many more neurons than nucleotides in the genome [4].

To paraphrase [4], part of the genetic code (genetic information) is converted into the nervous system during biological development. The genetic information drives the cell division, enabling to encode highly complex systems with a compact code. For example, a human brain contains approximately 10^{11} neurons, each connected to 10^5 other neurons (on average). If the connectivity graph was encoded by a destination list for each neuron, it would consume $10^{11} * 10^5 * \ln_2 10^{11} = 1.7 * 10^{17}$ bits of storage, whereas the number of genes in the human genome is of the order of $2 * 10^9$. [4] mentioned that the two numbers differ by more than 10 orders of magnitude, and marveled at the degree of compression achieved by the neuron developmental process.

As a simple example, Figure 2 illustrates the construction process of one branch of an ANN. Suppose that starting with the axiom ($\alpha = .$) and applying the second production rule $. \rightarrow f$ to the axiom, the resulting string is f. Applying the third rule (3.1) $f \rightarrow [f$ to string f yields a new string, $[f$. After one, two, and three applications of the fourth rule $[\rightarrow [Ff]$ on string $[f$, the strings become $[Ff]f$, $[Ff]Ff]f$, and $[Ff]Ff]Ff]f$, respectively. The last string, which is stored for later

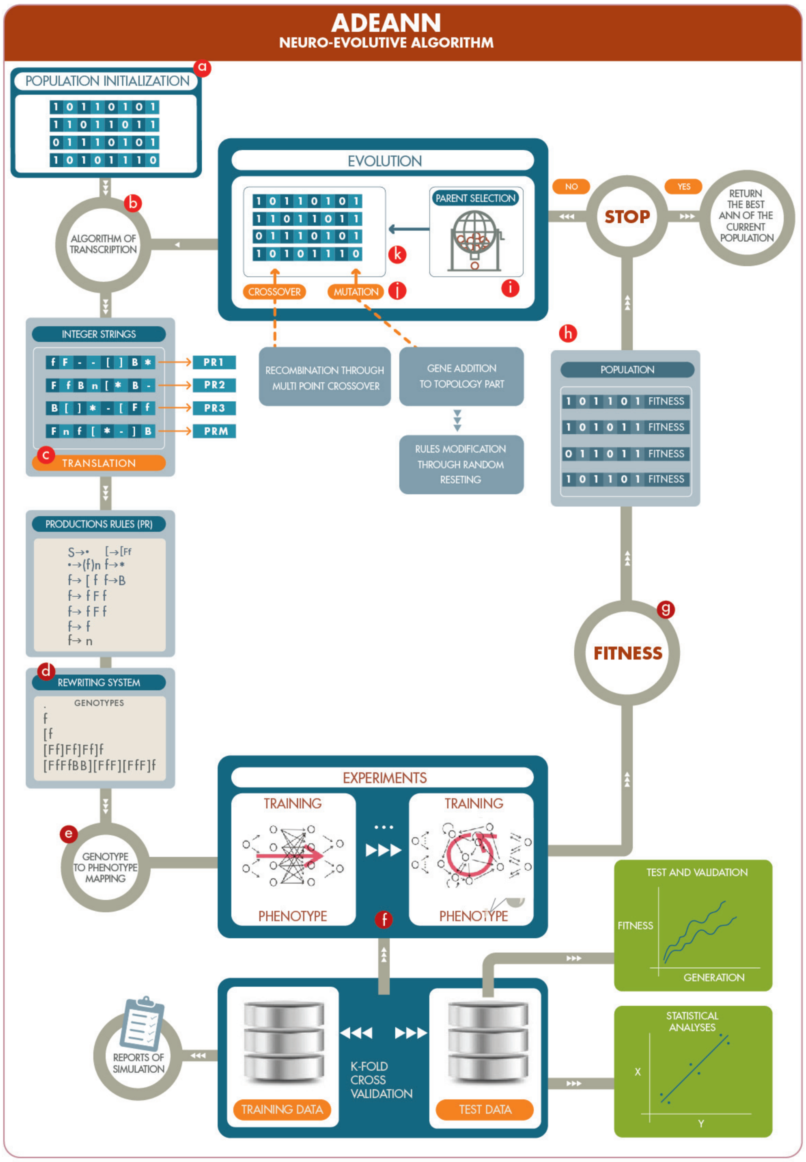


Figure 1. ADEANN links three computational paradigms: GA, an L-system, and an ANN. The evolutionary processes are defined as temporal genetic changes in the hypothetical DNAs, regulated by a selection mechanism. The hypothetical DNA of each individual encodes a set of production rules of an L-system, which coordinates the growth of artificial neurons. Each ANN is trained to solve a given problem and is thereafter tested to check its generalization or prediction capability.

Rule Identifier	Rule
1, 2	$S \to .$ (axiom) (2) $. \to (f \ldots f)n$
3	(3.1) $f \to [f$ (3.2) $f \to fFf$ (3.3) $f \to fF$ (3.4) $f \to n$
3, 4, 5	(3.5) $f \to f$ (3.6) $f \to fB$ (4) $[\to [Ff]$ (5) $f \to f*$

Table 1. The production rules of the parametric L-System with memory

use, represents a branch with three neurons (see last row, third column of Figure 2). Rule (5), $f \to f*$ (which means $f \to f[Ff]Ff]Ff]$, where $*$ denotes that the string is recovered from storage), is applied to the first f of the previous string $[Ff]Ff]Ff]f$. The resulting string is $[Ff[Ff]Ff]Ff]]Ff]Ff]f$, representing the phenotype shown in the fourth row and third column of Figure 2. This phenotype begins a new branch from N3. Here, we have used the principle of *reuse phenotypic structure*, which produces a variation of an existing structural theme.

Applying Rule (3.2) to the second f of the previous string, the resulting string is $[Ff[FfFf]Ff]Ff]]Ff]Ff]f$, with phenotype shown in the third row and third column of Figure 2. This phenotype creates a new neuron $N8$. The development continues by recursively iterating Rules (5) and (3.2). First, Rule (5) is applied to the sixth f of the previous string $[Ff[FfFf]Ff]Ff]]Ff]Ff]f$, yielding $[Ff[FfFf]Ff]Ff]]Ff]Ff]Ff]]Ff]f$. Consequently, a new branch begins from $N4$. Here, we have used the second principle of *reuse*, which creates separate developmental pathways from the same gene. Second, Rule (3.2) is applied to the seventh f of the previous string, yielding $[Ff[FfFf]Ff]Ff]]Ff]FfF]Ff]]Ff]f$. This step creates a synapse from $N41$ to $N8$. Figure 2 also shows the mapping between genotypes and phenotypes. Our indirect encoding method, presented in Section 4.2, generates ANNs with variable numbers of neurons in the range $[X, Y]$, where X and Y are computed by Equations (1) and (2), respectively. The classes of admissible ANNs generated by the L-Systems are direct and recurrent neural networks, with the recurrent output layer. The encoding rule is described by the rules presented in Table 1.

$$X = [N_{inputs} + N_{inputs} * random(NR_{min}) + N_{outputs}], \tag{1}$$

$$Y = [N_{inputs} + N_{inputs} * random(NR_{max}) + N_{outputs}]. \tag{2}$$

Here, N_{inputs} and $N_{outputs}$ are the numbers of neurons in the input and output layers of the ANN, respectively, and NR is the number of neurons at each ramification.

In a dismembered way, Figure 3 a) illustrates the iterative generation of ANN at each stage of the development process. The algorithm starts by interactively finding the placements of the hidden neurons from the inputs and their connections. It then determines the placements of the other hidden neurons and their connections. Finally, it determines the placements of the output neurons from the hidden neurons and their connections. The ANN construction process is illustrated in Figure 3 b). The entire developmental process follows an important biological property called the

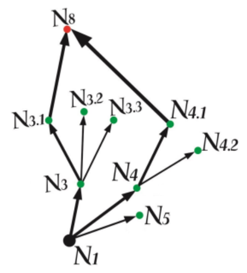
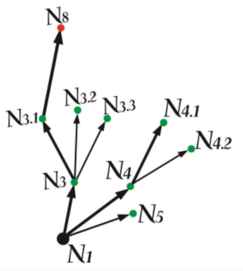
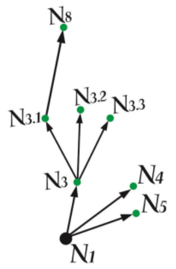
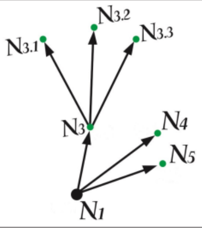
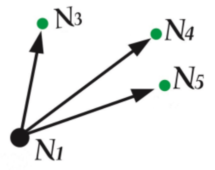
RULES	GENOTYPE	PHENOTYPE	IT
R3.2	n_3 $n_{3.1}$ n_6 $n_{3.2}$ $n_{3.3}$ n_4 $n_{4.1}$ n_6 $n_{4.2}$ n_5 n_1 [F *f* [F *f* F *f*]F *f*]F *f*]]F *f* [F *f* F]F *f*]]F *f*]*f* E_1 E_3 E_3 E_3 E_3E_1 E_4 E_4 E_4E_1 E_1	 N8 N3.1 N3.2 N3.3 N4.1 N4.2 N3 N4 N5 N1	IT.10
R5	n_3 $n_{3.1}$ n_8 $n_{3.2}$ $n_{3.3}$ n_4 $n_{4.1}$ $n_{4.2}$ n_5 n_1 [F *f* [F *f* F *f*]F *f*]F *f*]]F *f* [F *f*]F *f*]]F *f*]*f* E_1 E_3 E_3 E_3 E_3E_1 E_4 E_4 E_4E_1 E_1	 N8 N3.1 N3.2 N3.3 N4.1 N4.2 N3 N4 N5 N1	IT.9
R3.2	n_3 n_{31} n_8 n_{32} n_{33} n_4 n_5 n_1 [F *f* [F *f* F *f*]F *f*]F *f*]]F *f*]*f*]*f* E_1 E_3 E_3 E_3 E_3E_1 E_1 E_1		IT.8
R5	n_3 n_{31} n_{32} n_{33} n_4 n_5 n_1 [F *f* [F *f*]F *f*]F *f*]]F *f*]F *f*]*f* E_1 E_3 E_3 E_3 E_3E_1 E_1 E_1		IT.7
R4(3x) R3.1 R2 axiom	n_3 n_4 n_5 n_1 [F *f*]F *f*]F *f*]*f* E_1 E_1 E_1 E_1		IT.6

Figure 2. A simple example of the construction process of a branch of an iterated ANN using the rules of the L-system is illustrated in Table 1

hierarchical principle (recursive composition of sub-structures). In the topologies generated at each developmental stage, only those nodes forming paths to an input and output neuron are retained. The search through the search space is restricted to functional ANN topologies.

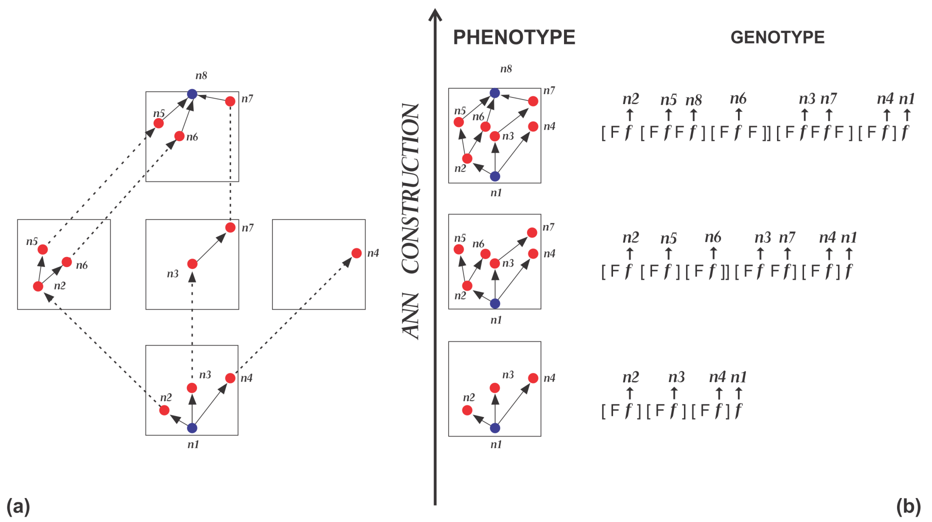

PHENOTYPE
GENOTYPE
ANN CONSTRUCTION
n1
n2
n3
n4
n5
n6
n7
n8
[F f [F f F f] [F f F]] [F f F f F] [F f] f
[F f [F f] [F f]] [F f F f] [F f] f
[F f] [F f] [F f] f
(a)
(b)

Figure 3. Construction of an iterated network

4.2 Rule Extraction by Genetic Algorithms

This subsection is dedicated to rule extraction by GAs. The neurons generated in the previous subsection are developed after the following process. To formulate a biologically realistic GA, we let the genes of the chromosomes (sequences of hypothetical DNA) encode a recipe (the production rules of the L-system described in Section 4.1 and illustrated in Table 1). The recursive rules in Table 1 drive the developmental stages of the neurons (Figure 2).

[40] argued that biological genes are meaningful only when translated into proteins following certain rules of growth for organ development. He emphasized that the complete genetic formula of neuronal development is unknown to geneticists.

In biological genetic processing (Figure 4 b)), DNA is transcribed into ribonucleic acid (RNA), and the RNA is translated into proteins. The proteins are derived from linear sequences of amino acids encoded by codons (groups of three nucleotides selected among U, C, A, and G) of the genetic code (Table 2). Table 2 specifies the translation products of different codons; for example, the codon UUU is translated into phenylalanine (Phe), and UUA is translated into leucine (Leu). In Figure 4 b), the protein is formed by a sequence of amino acids starting with methionine (Met)

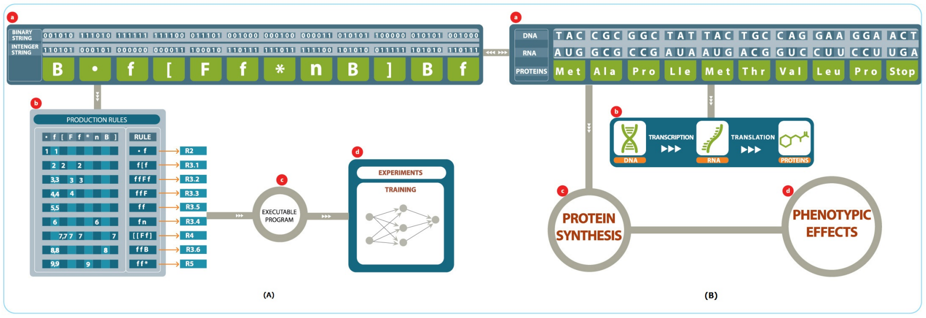


Figure 4. b) DNA transcription into RNA and translation of RNA into protein. a) In the analogous artificial process, a binary string is transcribed into an integer string and the string is translated into the production rules of the L-system.

and ending with proline (Pro). Such protein synthesis triggers all stages of the neuronal development (phenotypic effects), as shown in Figure 4 b).

The elements of the alphabet $\Sigma = \{., f, F, n, [,], *, B, \}$ of the L-system, described in Section 4.1 and displayed in bold font in Table 2, are a metaphor of the genetic code. Each two-bit sequence represents one nucleotide; for example, the set $(00, 01, 10, 11)$ symbolizes $(\mathrm{U}, \mathrm{C}, \mathrm{A}, \mathrm{G})$ in the original genetic code. Accordingly, six bits represent three nucleotides; that is, $(000000, 011111)$ symbolizes $(\mathrm{UUU}, \mathrm{CGG})$. Below we present the function Rule-Extraction-with-GA, which attempts to mimic DNA transcription and RNA translation as discussed in the previous paragraph.

Steps 1 and 2 of the function Rule-extraction-with-GA mimic the DNA transcription process into RNA, as shown in Figure 4 b). These steps are repeated for all individuals in the population. All the integer strings obtained after step 4 of this process see Figure 4 a) are stored in the array 'D' and evaluated in step 5. Step 5 translates the integer string to valid production rules of the L-system (Table 1) proposed in Section 4.1. Steps 5, 5.1 and 5.2 mimic the translation process of RNA into protein. The above steps are repeated for all rows in table 'D'.

Figure 4 a) illustrates the rule extraction for a single individual of the population. In this figure, the transcription string yields the integer string B.f[Ff.nB]Bf. We seek the shortest string containing all valid rules; in this case, $.f[Ff*nB]$. Steps 5, 5.1, and 5.2 identify the minimum substring $.f[Ff*nB]$ within the larger string $B.f[Ff.nB]Bf$. After finding the minimum string, we identify the positions at which the rules were found (see Figure 4 a)). For example, Rule 2 $(. \to f)$, symbolically represented by $(.f)$, is found at positions 1 and 2 of the string $.f[Ff*nB]$. Rule 3.1 $f \to [f$ is found at positions 1 and 2, and 3 and 5.

Collectively, these rules form a kind of executable program that prepares the artificial neurons for development. Once a set of valid production rules has been determined for each line of table 'D', the neuronal development presented in Section 4.1 is initiated. The development process is illustrated in Figure 2. The resulting

	00 (U)	01 (C)	10 (G)	11 (A)	
00 (U)	f (UUU)	F (UCU)	n (UAU)	. (UGU)	00 (U)
00 (U)	n (UUC)	. (UCC)	f (UAC)	F (UGC)	01 (C)
00 (U)	F (UUA)	f (UCA)	B (UAA)	f (UGA)	10 (A)
00 (U)	[(UUG)	n (UCG)	[(UAG)	$*$ (UGG)	11 (G)
01 (C)	f (CUU)	] (CCU)	n (CAU)	$*$ (CGU)	00 (U)
01 (C)	$*$ (CUC)	F (CCC)	f (CAC)	F (CGC)	01 (C)
01 (C)	] (CUA)	f (CCA)	$*$ (CAA)	[(CGA)	10 (A)
01 (C)	f (CUG)	$*$ (CCG)	B (CAG)	] (CGG)	11 (G)
10 (A)	$*$ (AUU)	] (ACU)	n (AAU)	f (AGU)	00 (U)
10 (A)	f (AUC))	B (ACC)	f (AAC)	B (AGC)	01 (C)
10 (A)	F (AUA)	[(ACA)	B (AAA)	n (AGA)	10 (A)
10 (A)	$*$ (AUG)	f (ACG)	$*$ (AAG)	] (AGG)	11 (G)
11 (G)	] (GUU)	[(GCU)	F (GAU)	n (GGU)	00 (U)
11 (G)	n (GUC)	B (GCC)	[(GAC)	. (GGC)	01 (C)
11 (G)	f (GUA)	] (CGA)	B (GAA)	F (GGA)	10 (A)
11 (G)	B (GUG)	f (GCG)	$*$ (GAG)	[(GGG)	11 (G)

Table 2. The genetic code from the perspective of mRNA, translated as in Figure 4 b). In the same table, the DNA's metaphor.

population of ANNs is trained and the fitness of each ANN is calculated by Equation (5). The ordered sequences of alphabetical characters (valid production rule: $.f[Ff*nB]$) are analogous to the sequences of amino acids that join into a protein (Figure 4 b)). Protein synthesis triggers all stages of biological development. As the function Rule-Extraction-with-GA begins the string reading from different positions and in both directions, the implicit parallelism level of the GA increases, alleviating the scalability problem.

4.3 Fitness Evaluation and Selection Mechanism

The fitness of an ANN can be appropriately measured by the mean square error (MSE) given by Equation (3). The MSE measures the expected squared distance between the predicted and true values. As such, it measures the quality of a predictor. The system adjusts the weights of its neural networks to minimize the MSE in the training and test sets. ADEANN aims not only to minimize the MSE of the ANN in the training set but also to generalize and properly predict in the test set. In Equation (3), q is the number of test samples. The fitness function selects the most suitable neural networks (direct or recurrent) for a specific class of problem, taking into consideration, that dynamical systems are more precisely estimated by recurrent neural networks (RNNs). In other words, depending on the simulated problem class, the fitness function, selects a direct or recurrent ANN most suitable for a specific task. The fitness function Equation (5) automatically implements a penalty approach that rewards economical ANNs with better generalization capacities and ease of implementation (see the last fifteen lines and second column of Table 5). This function selects networks by two criteria; the number of hidden-layer neurons and the MSE. Therefore, smaller networks score higher fitness values than larger networks with the same performance. RNN-evolving NEAs are rarely reported in the literature. Our ADEANN system partially fills this gap, and provides accurate, automatic direct and recurrent ANNs for pattern recognition problems and dynamical systems simulations.

Function Rule-Extraction-with-GA

Step 1 – Dynamically allocate an array of integers 'B' with dimension $[I \times G]$, where I is the number of individuals in the population and G is the number of desired genes. Randomly assign 0 or 1 to each position (i, g) in the array.

Step 2 – Obtain the complementary array 'B'.

```
For $i = 1$ to $G$:
For $g = 1$ to i
if $B[i, g] == 0$ $B[i, g] = 1$ else $B[i, g] = 0$;
```

Step 3 – Dynamically allocate an array of characters 'D' with dimension $[I \times (G/6)]$, where I is the number of individuals in the population and G is the number of desired genes.

Step 4 – For each chromosome (individual) of the population (each line of the binary array 'B'), read six-bit sequences, starting from the first one. Each group of six bits must be converted to a symbol of the alphabet defined by the grammar (L-system) described in Section 4.1. This conversion, which must obey Table 2, generates a string. To determine the character encoded by each six-bit sequence, Table 2 is read in the following order:

1. Determine which of the four rows on the left corresponds to the first two bits of the string;
2. choose the column matching the central two bits;
3. choose the row on the right corresponding to the last two bits.

For example, the strings 001000, and 111111 translate into characters $*$ and [, respectively. To each position (i', g') of the array 'D', assign the values converted from the binary characters in this step. Each line in array 'D' is represented by an integer string, as illustrated in Figure 4 a).

Step 5 – For each (row) of Table 'D' obtained in Step 4, find the substring encoding the valid production rules, reading each row from the first character and continuing along contiguous or non-contiguous positions.

Step 5.1 – Discard unnecessary characters until the substring $.f[Ff * nB]$ is obtained.

Step 5.2 – Repeat Step 5, beginning the read of the current line of array 'D' from other positions; for example, from the start to the end and vice versa. Individuals in the population with invalid rules are assigned zero fitness.

The fitness of an ANN can be appropriately measured by the mean square error (MSE) given by Equation (3). The MSE measures the expected squared distance between the predicted and true values. As such, it measures the quality of a predictor. The system adjusts the weights of its neural networks to minimize the MSE in the training and test sets. ADEANN aims not only to minimize the MSE of the ANN in the training set but also to generalize and properly predict in the test set. In Equation (3), q is the number of test samples. The fitness function selects the most suitable neural networks (direct or recurrent) for a specific class of problem. In other words, depending on the simulated problem class, the fitness function, selects a direct or recurrent ANN most suitable for a specific task.

The fitness function (Equation (5)) automatically implements a penalty approach that rewards economical ANNs with better generalization capacities and ease of implementation (see the last fifteen lines and second column of Table 5). This function selects networks by two criteria; the number of hidden-layer neurons and the MSE. Therefore, smaller networks score higher fitness values than larger networks with the same performance. RNN-evolving NEAs are rarely reported in the literature. Our ADEANN system partially fills this gap, and provides accurate, automatic direct and recurrent ANNs for pattern recognition problems and dynamical systems simulations.

In this paper, the reproducing individuals are selected by tournament selection, one of the most widely used selection strategies in GAs [12]. In tournament selection, the mating pool consists of tournament winners. The average fitness is usually higher in the mating pool than in the population. The fitness difference between the mating pool and the population reflects the selection intensity (I), given by Equation (6). The approximate solution is given by Equation (7), where x and y are the fitness values of the population and t is the tournament size. Tournament selection is expected to improve the fitness of each subsequent generation [31].

$$MSE(k) = \frac{\sum_{p=1}^{q}(d_p - o_{kp})^2}{q} \tag{3}$$

where d_p and o_{kp} are the desired output and the output of individual k for pattern p, respectively. In our system, we propose a penalty approach that prevents the algorithm from unnecessarily producing ANNs with a large number of hidden neurons. To this end, our system compares two fitness functions, given by Equations (4) and (5):

$$f1 = \left(\frac{A1}{MSE} + \frac{A2}{NNHL}\right), \tag{4}$$

$$f2 = \left[\exp(-MSE) \times \exp(-NNHL) + \frac{1}{(MSE \times NNHL)}\right] \tag{5}$$

where MSE is the mean squared error and NNHL is the number of neurons in the hidden layer.

$$I = \int_{-\infty}^{\infty} tx \frac{1}{2\pi} e^{\frac{-x^2}{2}} \left(\int_{-x}^{\infty} \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} \mathrm{d}y\right)^{t-1} \mathrm{d}x, \tag{6}$$

$$I \approx \sqrt{2\left(\ln(t) - \ln\left(\sqrt{4.44\ln(t)}\right)\right)}, \tag{7}$$

$$LD(t) = t^{\frac{-1}{t-1}} - t^{\frac{-t}{t-1}}. \tag{8}$$

Table 3 shows how the selection intensity calculated by Equation (8) varies with tournament size.

Tournament Size (t)	1	2	3	5	10	30
Selection Intensity (I)	0	0.56	0.85	1.15	1.53	2.04

Table 3. Relation between tournament size and selection intensity

In tournament selection, larger tournament size t is lead to higher expected loss of diversity (LD) [31]. As evident from Equation (8), approximately 50 % of the population is lost at tournament size $t = 5$. Diversity is important in GAs because crossing over a homogeneous population does not yield new solutions.

In tournament selection, larger tournament size t is lead to higher expected loss of diversity (LD) [31]. Diversity is important in GAs because crossing over a homogeneous population does not yield new solutions. Accordingly, the tournament size t of ADEANN is set to 3, meaning that three chromosomes compete with each other, and the best chromosome among these three is selected for reproduction.

By recombining the genetic codes of two parents, the crossover operator produces two solutions in a yet unvisited region of the search space. For each pair of parents, the crossover operator is applied with a certain crossover probability (Pc). The benefits of increasing the number of crossover points in the crossover operation have been reported in many (largely empirical) studies [36]. ADEANN crosses multiple randomly selected points in the chromosome. The number of crossover points is proportional to the chromosome length divided by six and multiplied by the crossover rate (Cr). After applying the crossover operator, the genes of each resulting offspring are altered by the mutation operator. The mutation operator allows the GA to explore new solution space avoiding trapping in local optima. The mutation points in

ADEANN are randomly chosen with probability Pm. Their number is proportional to the chromosome length divided by six and multiplied by the mutation rate (Mr).

4.4 Network Models and Their Biological Relevance

The first network model generated by ADEANN in the classification experiment was a single hidden-layer feed-forward network. This model, characterized by a three-layer network of simple processing units connected by acyclic links, is commonly employed in systems biology studies [37], including evolutionary studies. Information flows through the ANN in discrete time. The output o_j of node j is calculated by Equation (9):

$$\vartheta(o_j) = \frac{1}{1 + \exp^{-k.\sum_{i \epsilon I_j}(w_{ij}x_i + b_j)}}. \tag{9}$$

Here, I_j is the set of nodes connected to node j, w_{ij} is the strength of the connection between node i and node j, o is the output value of node i, and b_j is the bias. The parameter k measures the steepness of the sigmoidal function (Equation (9)). As k is positive, the sigmoidal function is monotonically increasing, continuous and differentiable over the whole domain. In our experiments, we assigned the typical value for the training of neural networks with BP (namely, $k = 1$).

The parameters of the neural network w_{ij} are changed by an amount Δw_{ij}, which is calculated by Equation (10):

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{10}$$

where the parameter η is the learning rate and E is the error in the output layer. The δ term in Equation (11) is a momentum term, introduced by [38] to accelerate the learning process while avoiding instability in the algorithm.

The second network model generated by ADEANN, which was employed in the time-series forecasting experiments, was the single hidden-layer recurrent network. [39] discussed the nature of recurrence within the neocortex, a part of the brain that processes sensory stimuli. Such a recurrent structure of biological neuronal circuits is relevant to the field of artificial intelligence.

$$\Delta w_{ij}(t+1) = -\eta \frac{\partial E}{\partial w_{ij}} + \delta \Delta w_{ij}(t). \tag{11}$$

5 EXPERIMENTAL SETUP AND RESULTS WITH ADEANN

In this section we evaluate the performance of the ADEANN in evolving the architectures of ANNs.

5.1 Experimental Setup

Before conducting the formal experiments, we must set the ADEANN parameters. The parameter values were empirically determined from the results of several XOR experiments. Experiments were run on a V14T-5470-A60 Core i7 computer with 8 GB memory, operating at 3.2 GHz. The various combinations of the parameter values are listed in Table 5. The first column specifies the numbers of the experiments, generations, and individuals per population. The second column states the neural network solution returned by the search process. The third, fourth, and fifth columns, respectively, specify the chromosome length, the number of crossover and mutation points, and the crossover rate. The last four columns present the simulations results: the average (μ) and standard deviation (σ) of the number of neurons in the hidden layer, the MSE in the generalization phase, and the percentage of production rules. The superior settings are given in Table 4.

Pop. size (v)	Generations	C_r	P_m	P_{elit}
[30–100]	[50–500]	$[0.3, 0.9]$	0.1	2 %

Table 4. Parameters used in the experiments

The results of the XOR problem after twenty experiments are summarized in Table 5. In the last fifteen simulations, the fitness given by Equation (5) automatically prevents the algorithm from unnecessarily producing complex ANNs with a large number of hidden neurons, and encourages smaller topologies. This constraint improves the generalization capacity of the ANNs. All networks returned by the search process contained two neurons in the input layer, two in the hidden layer, and one in the output layer, and can be described by ANN $= (2, 2, 1)$. This structure is not guaranteed by the fitness function Equation (4), because the appropriate coefficients for this equation are determined by a problem dependent, time-consuming process that obtains A1 and A2 by trial-and-error. Furthermore, there is no guarantee that the search will return a small network. In the first five experiments (summarized in the first five lines of Table 5), all of the ANNs returned by the search process contained 8, 7, 6, or 5 neurons in the hidden layer. The results of the best (19th) and worst (first) experiments, in terms of small MSE and minimum number of hidden neurons, are highlighted in bold in Table 5.

From Table 6, we observe that in 100 runs NEAT [6] finds a structure for XOR in 32 generations (4 755 networks evaluated, $\sigma = 2.553$). The average number of hidden nodes in the solution network was $\mu = 2.35$. The 13th experiment (Table 6) yielded the closest solution, with $\mu = 3.31$ and $\sigma = 1.65$ in 50 generations. ADEANN has a similar average classification accuracy to DENN [32], but more accurately estimates the best network in the search process (MSE $\leq$ 0.000024). ADEANN easily solves the XOR problem while maintaining a small architecture. The numbers of nodes and connections were close to optimal, considering that the smallest ANN had a single hidden unit. Moreover, the ADEANN always found a solution. The total number of ANNs trained in the experiments was 30 100.

GxP	ANN	CL	(c, m)	Cr	μ	σ	MSE	PR [%]
E1-50x30	**(2, 8, 1)**	**180**	**(9, 3)**	**0.6**	**6.50**	**1.50**	**0.000046**	6.67
E2-50x30	(2, 7, 1)	216	(10, 3)	0.6	4.80	2.32	0.000049	16.67
E3-50x30	(2, 5, 1)	252	(12, 4)	0.6	4.80	1.72	0.000048	25
E4-60x30	(2, 6, 1)	252	(12, 4)	0.9	3.86	1.81	0.000032	23.33
E5-50x20	(2, 6, 1)	276	(13, 5)	0.6	6.00	1.94	0.000029	20.28
E6-50x30	(2, 2, 1)	180	(18, 3)	0.3	5.67	2.62	0.000076	9
E7-50x30	(2, 2, 1)	180	(18, 3)	0.6	3.20	0.75	0.000075	13.40
E8-50x30	(2, 2, 1)	180	(18, 3)	0.9	4.50	2.18	0.000051	11.33
E9-50x20	(2, 2, 1)	216	(32, 28)	0.6	6.00	2.07	0.000056	22.33
E10-50x20	(2, 2, 1)	216	(32, 28)	0.9	3.71	2.43	0.000056	21.53
E11-50x30	(2, 2, 1)	252	(37, 33)	0.6	3.89	3.65	0.000056	28.67
E12-60x30	(2, 2, 1)	252	(37, 33)	0.9	4.36	1.72	0.000055	26.73
E13-50x30	(2, 2, 1)	300	(45, 40)	0.6	3.31	1.65	0.000053	51.20
E14-60x30	(2, 2, 1)	300	(45, 40)	0.9	3.77	2.01	0.000053	41.93
E15-50x30	(2, 2, 1)	360	(54, 48)	0.6	3.69	2.14	0.000056	50.73
E16-50x30	(2, 2, 1)	360	(54, 48)	0.9	3.71	1.87	0.000056	54.33
E17-50x30	(2, 2, 1)	390	(58, 52)	0.6	3.79	1.61	0.000053	63.33
E18-50x30	(2, 2, 1)	390	(58, 52)	0.9	5.10	2.05	0.000053	63.40
E19-100x30	**(2, 2, 1)**	516	**(77, 68)**	**0.9**	**5.28**	**2.03**	**0.000024**	96.67
E20-100x30	(2, 2, 1)	516	(77, 68)	0.6	4.15	2.17	0.000025	90

Table 5. Genetic algorithm parameters and simulation results of 20 experiments of the XOR problem. ANN = (X, Y, Z), where X, Y, and Z are the number of neurons in the input, hidden and output layers, respectively.

Problem	NEAT	ADEANN	DENN
XOR	– **2.35**	98.84 %[a] 3.31[b]	**98.97** % –
	2.553 –	**1.65**[c] **0.000024**[d]	– 0.004865

Table 6. Comparison between ADEANN and other methods in the XOR problem. [a] Average classification accuracy, [b] Average number of hidden neurons, [c] Standard deviation of the number of hidden neurons, [d] MSE, DENN [32].

5.2 Consumer Price Index – CPI Problem

To verify the viability of the method for large problems, we carried out experiments for the prediction of the time series of the consumer price index (CPI), for the period of January 1998 to May 2010 (source: Central Bank of Brazil, as seen in Table 7). The CPI quantifies the costs of products at different moments. In other words, it measures changes through time in the price level of consumer goods and services purchased by households and is useful for the calculation of the inflation rate.

In five runs, the best experiment shows that the ADEANN system finds an ANN for CPI in 50 generations (1 500 networks evaluated, std σ = 4.39) and MSE =

	Jan	Feb	Mar	Apr	May	Jun
1998	0.0126	0.0014	0.0033	0.0023	0.0014	0.0041
1999	0.0064	0.0141	0.0095	0.0052	0.0008	0.0065
2000	0.0101	0.0005	0.0051	0.0025	0.004	−0.0001
2001	0.0064	0.004	0.0056	0.0086	0.0041	0.0052
2002	0.0079	0.0014	0.0042	0.0071	0.0028	0.0055
2003	0.0232	0.0137	0.0106	0.0112	0.0069	−0.0016
2004	0.0108	0.0028	0.0046	0.0031	0.0071	0.0078
2005	0.0085	0.0043	0.007	0.0088	0.0079	−0.0005
2006	0.0065	0.0001	0.0022	0.0034	−0.0019	−0.004
2007	0.0069	0.0034	0.0048	0.0031	0.0025	0.0042
2008	0.0097	0.0056	0.0045	0.0072	0.0087	0.0077
2009	0.0083	0.0021	0.0061	0.0047	0.0039	0.0012
2010	0.0129	0.0068	0.0086	0.0076	0.0021	–

	Jul	Aug	Sep	Oct	Nov	Dec
1998	−0.0025	−0.0052	−0.0017	0.002	−0.0019	0.0009
1999	0.012	0.0048	0.0019	0.0092	0.0112	0.006
2000	0.0191	0.0086	0.0004	0.0002	0.004	0.0062
2001	0.0136	0.0054	0.0012	0.0071	0.0085	0.007
2002	0.0103	0.0076	0.0066	0.0114	0.0314	0.0194
2003	0.0034	0.0013	0.0076	0.0021	0.0033	0.0043
2004	0.0059	0.0079	0.0001	0.001	0.0037	0.0063
2005	0.0013	−0.0044	0.0009	0.0042	0.0057	0.0046
2006	0.0006	0.0016	0.0019	0.0014	0.0024	0.0063
2007	0.0028	0.0042	0.0023	0.0013	0.0027	0.007
2008	0.0053	0.0014	−0.0009	0.0047	0.0056	0.0052
2009	0.0034	0.002	0.0018	0.0001	0.0026	0.0024

Table 7. Normalized values for CPI in the period of January 1998 to May 2010. Source: Central Bank of Brazil.

0.000015972. On average, a solution network has $\mu = 5.21$ hidden nodes. The worst performance takes 1 500 evaluations, or about 50 generations, the standard deviation was $\sigma = 5.42$, the average $\mu = 9.78$ for the number of hidden nodes and MSE $= 0.0000159912$. Since the fitness function (5) takes into account the minimal number of neurons in the hidden layer (NNHL) and a minimal mean squared error (MSE). ADEANN solves the CPI problem without trouble. The lowest ANN for that problem has the following specification: $(1, 3, 1)$. The largest ANN obtained for the CPI problem has the specification $(1, 18, 1)$, i.e., with 18 neurons in the hidden layer. It had an MSE $= 0.000015733$ and good generalizability, with MSE ≤ 0.0001 and is better than the ANN $(1, 3, 1)$. However, as the fitness, as given by Equation (5), favors the smaller ANNs with good generalization capacity, the ANN $(1, 3, 1)$ is ranked the best.

5.3 Prediction of the Effect of a New Drug on Breast Cancer (BC)

The problem of predicting the effect of a new drug on breast cancer [26] is to find an analogous mapping between the set of 15 experimentally induced tumors and a clinical tumor (malignant breast cancer), based on their known reactions to the same drugs, so that we are able to predict the effect of a new drug on the clinical tumor after having studied the experimental tumors. The solution is based on the presumed similarity between this tumor and a set of experimental tumors. Table 8 illustrates the dataset used for training, which contains the known effect of 26 established drugs on 15 experimental tumors and on a clinical tumor.

	T1	2	3	4	5	6	7	8	9	T0	11	12	13	14	15	TC
D1	1	1	0.5	0.5	1	0.5	1	1	0.5	0.5	0.5	1	1	1	0.5	1
2	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1
3	1	1	1	1	0	0	1	1	0	1	0	1	1	1	1	0
4	1	1	0.5	0	0	0.5	1	0.5	0.5	0.5	0.5	1	1	0.5	0.5	1
5	1	1	1	0	1	0	1	1	0	1	0	0	1	0	1	1
6	1	1	0	1	0	0.5	1	1	1	0	0	0	1	1	0.5	1
7	1	1	0	1	0	0.5	1	1	1	0	0	0	1	1	0.5	1
8	0	1	0	1	0	0.5	0	1	1	0	0	1	0	0.5	1	1
9	1	1	1	0	0	0	0	0	1	0	0.5	1	0	0	1	1
10	1	1	1	1	1	0	1	1	1	0	1	0	0	0	1	1
11	1	1	1	0	0	0.5	0	0	0.5	0	0.5	1	1	0	0.5	1
12	1	0.5	1	0.5	0	0.5	0	0	0.5	0	1	0.5	1	0.5	0.5	1
13	0	0	1	0	1	0	0.5	0.5	0.5	0	0.5	1	1	0.5	0.5	0
14	1	1	1	0	0.5	0.5	0	0.5	0	0	0	0.5	0.5	0.5	0	0
15	1	0.5	1	1	0	0	1	1	0	0	0	0	01	0	0	0
16	1	0.5	0.5	1	1	1	0	0.5	1	1	0.5	0.5	1	0.5	1	1
17	1	0	0.5	1	1	0.5	1	0.5	0.5	1	0.5	0.5	0	0.5	0.5	1
18	1	0.5	1	0.5	0	0.5	1	1	0.5	0	0.5	1	1	0.5	0.5	1
19	0	0	1	1	0.5	1	1	1	0.5	0	0.5	0.5	1	0.5	0.5	1
20	1	1	1	0	0	0.5	0	1	0.5	0.5	0.5	0.5	1	1	1	0
21	1	1	1	1	0	0	1	0	0	1	1	0	1	1	1	1
22	1	0	0.5	1	0.5	1	0	0.5	1	1	0.5	0.5	1	0	1	1
23	1	0	0	1	1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0	0.5	0.5	1
24	1	0.5	0.5	0.5	0	0.5	1	0.5	0.5	0	0.5	1	0.5	0.5	0.5	1
25	0	0	0.5	0.5	0	0	1	1	1	0.5	0.5	1	0	1	1	1
26	0	0.5	0.5	0	0	0.5	1	0.5	1	0.5	0.5	0	0.5	0.5	1	0.5

Table 8. The effect of 26 drugs on 15 experimental tumors and on a tumor in a patient with breast cancer

In five runs, the best experiment shows that the ADEANN system finds an ANN for BC in 100 generations (3 000 networks evaluated, std $\sigma = 1.206$) and MSE $= 0.000005$. On average, a solution network has $\mu = 6.83$ hidden nodes. The minimal neural network which was found has the following specification (neurons in the input layer, neurons in the hidden layer, neurons in the output layer) = $(15, 6, 1)$. The testing results for that network are shown in Table 9. In the method of [26], on average a solution network had $\mu = 3.6$ neurons in the hidden layer, std $\sigma = 1.1$ and MSE = 0.015. Table 10 illustrates a comparison with a wider variety of methods.

Obtained Output	Desired Output	Error
0.999700	1.0	0.000000045
0.998717	1.0	0.000000823
0.999973	1.0	0.000000000
0.999963	1.0	0.000000001
0.999546	1.0	0.000000103
0.998482	1.0	0.000001153
0.996649	1.0	0.000005616
0.997020	1.0	0.000004441
0.007759	0.0	0.000030104
0.999862	1.0	0.000000000
0.999046	1.0	0.000000455

Table 9. Testing results for the ANN for predicting the effect of 15 new drugs on malignant breast cancer. Eleven test examples are used. If a threshold of 1.0 is chosen to decide on an active neuron, the effects of all the new drugs tested are correctly predicted.

Dataset	ADEANN	G3PCX [48]	GA [46]
BC	**99.11** %[a] 14.72[b] 2.54[c] **0.000115** %[d]	98.94 % 5 0 –	98.88 % 5 0 –

Table 10. Comparison between ADEANN and other methods for the Breast Cancer Dataset. [a] Average classification accuracy, [b] Average number of hidden neurons, [c] Standard deviaton of the number of hidden neurons, [d] MSE.

One of the major problems facing researchers is the selection of hidden neurons using neural networks. There is no theory to tell us how to choose the optimal number of hidden units, moreover, it cannot be generalized that accuracy will increase proportionally with increase in number of hidden neurons or vice versa. [43] suggests that the optimal number of neurons in the hidden layer for a 3-layer neural network is given by Equation (10), where N = number of inputs, m = number of outputs and NNHL is the number of neurons in the hidden layer. The proposed model improves the accuracy and minimal error. For this problem $m = 1$ and $N = 15$, replacing these values in Equation (12) yields NNHL = 11.2. Regarding Table 10, ADEANN indicates that the average number of hidden neurons for BC problem is 14.72. Which shows that ADEANN yielded the closest solution among all three methods.

$$\mathrm{NNHL} = \sqrt{(m+2)N} + 2 * \sqrt{\frac{N}{m+2}}. \tag{12}$$

6 DISCUSSION

The first challenge that we have addressed was the scalability problem. Our approach includes improvements to our previous proposal [7]. First, the chromosome used by the GA in our research has a variable length, from 180 to 516 bits, in contrast to before, when it had a fixed length of 1 024 bits. For this reason, our current approach evolves a compressed description of the ANN that reduces the search space of the GA. Furthermore, it reduces the scalability problem of other methods [27, 28]. In the section about the experiments we present results that confirm these improvements.

The method of [28], as described in [7], improved the scalability problem presented by the method of [27], since an ANN with N neurons will result in a binary string of $(N(N+1)/2)$ in contrast to the method of [27], which has a quadratic behavior N^2. If the two methods [27, 28] had used the chromosome length that we used in the experiments described in Section 5, which are in the interval $[180, 516]$, the methods of [27] and [28] would generate ANNs with a maximal number of neurons in the intervals $[13, 22]$ and $[18, 31]$, respectively. Depending on the value of NR, our L-System can generate larger ANNs than the methods of [27, 28].

Our method also enables network optimization. As discussed in Section 4.3, the fitness function (Equation (5)) automatically implements a penalty approach that rewards economical ANNs with better generalization capacities and ease of implementation (see the last fifteen lines and the second column of Table 5). This function selects networks by two criteria: the number of hidden-layer neurons, and the MSE. Therefore, smaller networks score higher fitness values than larger networks with the same performance. RNN-evolving NEAs are rarely reported in the literature. Our ADEANN system partially fills this gap, and provides accurate, automatic direct and recurrent ANNs for pattern recognition problems and dynamical systems simulations. Many methods do not worry, as stated by [24], about avoiding optimizing overly complex topologies. Thus, at least part of the evolutionary search will be conducted in an unnecessarily high dimensional space, which is not desirable, as in some situations, less complex topologies can simulate the same task with the same precision.

Our method also enables network optimization. As discussed in Section 5.1, the fitness function Equation (5) automatically implements a penalty approach that rewards economical ANNs with better generalization capacities and ease of implementation (see the last fifteen lines and second column of Table 5). This function selects networks by two criteria; the number of hidden-layer neurons and the MSE. Therefore, smaller networks score higher fitness values than larger networks with the same performance. The overall topology and the number of hidden neurons are indeed dependent on the configuration of the algorithm.

Methods that increase or decrease progressively the size of the hidden layer are relatively fast, but may generate sub-optimal solutions (all greedy optimisation methods are liable to converge to local optima). Metaheuristics (like genetic algorithms) are more likely to find the global optimum but might be too slow.

However, our proposed approach is not without weaknesses. Our approach is much more sensitive to the initial conditions and constants than existing methods. As the initial population is randomly initialized with 0s and 1s, the generated genotypes are difficult to control. Therefore, when configuring the initial population, we must apply stochastic methods such as the Gaussian distribution. Moreover, as the variable NR in Equations (1) and (2) is randomly generated, the algorithm cannot generate all possible neuron topologies in the interval $[X, Y]$ in a specific generation.

7 CONCLUSION

ADEANN constitutes an automatic system to simulate tasks of pattern recognition, without any previous knowledge from the user. Also, two modifications of an earlier method [7] have been evaluated. The first one involves the fitness function, given by Equation (5). It explicitly rewards smaller solutions. Our approach solves the three problems presented previously, in Section 5, without trouble in finding small topologies. The second, our L-System, presented in Section 4.2, enables generating more complex, direct, recurrent and multilayer networks, than the previous one [7]. Other contributions are: our L-System allows generating ANN architectures without requiring additional configuration of substrates for each particular type of problem, as required by the methods of [8, 10], our model automates this process. Only the method of [15] formalizes the problem of ADANNs as a local search strategy. In Section 3, we described a new approach to formalizing the problem of ADANNs as a local search based on rational agents. Our approach increases the level of implicit parallelism of GA, which evolves a compressed description of the L-rules and enables generating more complex, direct, recurrent and multilayer networks than the previous one [7].

NEAs that evolve recurrent neural networks are rarely found in the literature [34]. To partially fill this gap, we designed ADEANN as an accurate automatic method for designing direct and recurrent ANNs and thereby solving pattern recognition problems and simulating dynamical systems.

Ultimately, the biological inspiration addressed in our research, makes it original. The merits of our bio-inspired algorithm compared to conventional algorithms are: flexibility, performance, scalability, flexibility in decision making and improvement scope. In future work, we will tackle other challenges, such as partially connected networks and ANNs that use Hebbian learning, thus exploiting aspects of neural plasticity.

REFERENCES

[1] Stanley, K. O.—Miikkulainen, R.: A Taxonomy for Artificial Embryogeny. Artificial Life, Vol. 9, 2003, No. 2, pp. 93–130, doi: 10.1162/106454603322221487.

[2] Bukhtoyarov, V.—Semenkin, E.: Evolutionary Three-Stage Approach for Designing of Neural Networks Ensembles for Classification Problems. In: Tan, Y.,

Shi, Y., Mo, H. (Eds.): Advances in Swarm Intelligence (ICSI 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7928, 2013, pp. 467–477.

[3] Krömer, P.—Platoš, J.—Snášel, V.: Nature-Inspired Meta-Heuristics on Modern GPUs: State of the Art and Brief Survey of Selected Algorithms. International Journal of Parallel Programming, Vol. 42, 2014, No. 5, pp. 681–709, doi: 10.1007/s10766-013-0292-3.

[4] Gruau, F.: Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm. Unpublished doctoral dissertation, Ecole Normale Superieure de Lyon, l'Université Claude Bernard Lyon 1, 1994.

[5] Park, H. S.—Tran, N. H.: Development of a Biology Inspired Manufacturing System for Machining Transmission Cases. International Journal of Automotive Technology, Vol. 14, 2013, No. 2, pp. 233–240, doi: 10.1007/s12239-013-0026-y.

[6] Stanley, K. O.—Miikkulainen, R.: Evolving Neural Networks Through Augmenting Topologies. Evolutionary Computation, Vol. 10, 2002, No. 2, pp. 99–127, doi: 10.1162/106365602320169811.

[7] de Campos, L.—Roisenberg, M.—de Oliveira, R.: Automatic Design of Neural Networks with L-Systems and Genetic Algorithms – A Biologically Inspired Methodology. The 2011 International Joint Conference on Neural Networks (IJCNN), 2011, pp. 1199–1206, doi: 10.1109/IJCNN.2011.6033360.

[8] Gauci, J.—Stanley, K. O.: Autonomous Evolution of Topographic Regularities in Artificial Neural Networks. Neural Computation, Vol. 22, 2010, No. 7, pp. 1860–1898, doi: 10.1162/neco.2010.06-09-1042.

[9] Lee, D.-W.—Kong, S. G.—Sim, K.-B.: Evolvable Neural Networks Based on Developmental Models for Mobile Robot Navigation. Proceedings of 2005 IEEE International Joint Conference on Neural Networks (IJCNN '05), 2005, Vol. 1, 2005, pp. 337–342.

[10] Risi, S.—Stanley, K. O.: An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density, and Connectivity of Neurons. Artificial Life, Vol. 18, 2012, No. 4, pp. 331–363.

[11] Benardos, P. G.—Vosniakos, G.-C.: Optimizing Feedforward Artificial Neural Network Architecture. Engineering Applications of Artificial Intelligence, Vol. 20, 2007, No. 3, pp. 365–382, doi: 10.1016/j.engappai.2006.06.005.

[12] Miller, G. F.—Todd, P. M.—Hegde, S. U.: Designing Neural Networks Using Genetic Algorithms. Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989, pp. 379–384.

[13] Zhang, J.-R.—Zhang, J.—Lok, T.-M.—Lyu, M. R.: A Hybrid Particle Swarm Optimization-Back-Propagation Algorithm for Feedforward Neural Network Training. Applied Mathematics and Computation, Vol. 185, 2007, No. 2, pp. 1026–1037, Special Issue on Intelligent Computing Theory and Methodology.

[14] Zhang, L.—Sun, Y.: Evolved Neural Network Based Intelligent Trading System for Stock Market. In: Tan, Y., Shi, Y., Mo, H. (Eds.): Advances in Swarm Intelligence (ICSI 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7928, 2013, pp. 478–488.

[15] Donate, J. P.—Sanchez, G. G.—de Miguel, A. S.: Time Series Forecasting. A Comparative Study Between an Evolving Artificial Neural Networks System and Statistical Methods. International Journal on Artificial Intelligence Tools, Vol. 21, 2012, No. 1, Art. No. 1250010.

[16] Clune, J.—Mouret, J.-B.—Lipson, H.: The Evolutionary Origins of Modularity. Proceedings of the Royal Society B, Vol. 280, 2013, No. 1775, Art. No. 20122863, Cite arxiv:1207.2743, doi: 10.1098/rspb.2012.2863.

[17] Cantu-Paz, E.—Kamath, C.: An Empirical Comparison of Combinations of Evolutionary Algorithms and Neural Networks for Classification Problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B, Vol. 35, 2005, No. 5, pp. 915–927, doi: 10.1109/TSMCB.2005.847740.

[18] Deb, K.—Anand, A.—Joshi, D.: A Computationally Efficient Evolutionary Algorithm for Real-Parameter Optimization. Evolutionary Computation, Vol. 10, 2002, No. 4, pp. 371–395.

[19] Risi, S.—Stanley, K. O.: A Unified Approach to Evolving Plasticity and Neural Geometry. Proceedings of the International Joint Conference on Neural Networks (IJCNN 2012), IEEE, 2012, pp. 1–8, doi: 10.1109/IJCNN.2012.6252826.

[20] Lee, S.—Yosinski, J.—Glette, K.—Lipson, H.—Clune, J.: Evolving Gaits for Physical Robots with the HyperNEAT Generative Encoding: The Benefits of Simulation. In: Esparcia-Alcázar, A. I. (Ed.): Applications of Evolutionary Computation (EvoApplications 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7835, 2013, pp. 540–549.

[21] de Campos, L. M. L.—de Oliveira, R. C. L.: A Comparative Analysis of Methodologies for Automatic Design of Artificial Neural Networks from the Beginnings Until Today. Proceedings of the 2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence (BRICS-CCI-CBIC '13), IEEE Computer Society, Washington, DC, USA, 2013, pp. 453–458, doi: 10.1109/BRICS-CCI-CBIC.2013.81.

[22] Simon, H. A.: The Sciences of the Artificial. 3rd Edition, The MIT Press, Cambridge, MA, 1996.

[23] Russell, S. J.—Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[24] Whiteson, S.: Evolutionary Computation for Reinforcement Learning. In: Wiering, M., van Otterlo, M. (Eds.): Reinforcement Learning. Springer, Berlin, Heidelberg, Adaptation, Learning, and Optimization, Vol. 12, 2012, pp. 325–355, doi: 10.1007/978-3-642-27645-3_10.

[25] Prusinkiewicz, P.—Runions, A.: Computational Models of Plant Development and Form. The New Phytologist, Vol. 193, 2012, No. 3, pp. 549–569, doi: 10.1111/j.1469-8137.2011.04009.x.

[26] Islam, M.—Yao, X.—Murase, K.: A Constructive Algorithm for Training Cooperative Neural Network Ensembles. IEEE Transactions on Neural Networks, Vol. 14, 2003, No. 4, pp. 820–834.

[27] Kitano, H.: Designing Neural Networks Using Genetic Algorithms with Graph Generation System. Complex Systems, Vol. 4, 1990, pp. 461–476.

[28] Boozarjomehry, R. B.—Svrcek, W. Y.: Automatic Design of Neural Networks Structures. Computers and Chemical Engineering, Vol. 25, 2001, No. 7-8, pp. 1075–1088, doi: 10.1016/S0098-1354(01)00680-9.

[29] Sánchez, D.—Melin, P.: Optimization of Modular Granular Neural Networks Using Hierarchical Genetic Algorithms for Human Recognition Using the Ear Biometric Measure. Engineering Applications of Artificial Intelligence, Vol. 27, 2014, pp. 41–56, doi: 10.1016/j.engappai.2013.09.014.

[30] Kazaryan, D. E.—Savinkov, A. V.: Grammatical Evolution for Neural Network Optimization in the Control System Synthesis Problem. Procedia Computer Science, Vol. 103, 2017, pp. 14–19, doi: 10.1016/j.procs.2017.01.002.

[31] Oladele, R. O.—Sadiku, J. S.: Genetic Algorithm Performance with Different Selection Methods in Solving Multi-Objective Network Design Problem. International Journal of Computer Applications, Vol. 70, 2013, pp. 5–9.

[32] Ilonen, J.—Kamarainen, J.-K.—Lampinen, J.: Differential Evolution Training Algorithm for Feed-Forward Neural Networks. Neural Processing Letters, Vol. 17, 2003, No. 1, pp. 93–105.

[33] de Campos, L. M. L.—de Oliveira, R. C. L.—Roisenberg, M.: Optimization of Neural Networks Through Grammatical Evolution and a Genetic Algorithm. Expert Systems with Applications, Vol. 56, 2016, pp. 368–384, doi: 10.1016/j.eswa.2016.03.012.

[34] Hornby, G. S.—Pollack, J. B.: Creating High-Level Components with a Generative Representation for Body-Brain Evolution. Artificial Life, Vol. 8, 2002, pp. 223–246, doi: 10.1162/106454602320991837.

[35] Lee, D. W.—Seo, S.-W.—Sim, K.-B.: Evolvable Neural Networks Based on Developmental Models for Mobile Robot Navigation. International Journal of Fuzzy Logic and Intelligent Systems, Vol. 7, 2007, pp. 176–181, doi: 10.5391/IJFIS.2007.7.3.176.

[36] De Jong, K. A.—Spears, W. M.: A Formal Analysis of the Role of Multi-Point Crossover in Genetic Algorithms. Annals of Mathematics and Artificial Intelligence, Vol. 5, 1992, No. 1, pp. 1–26, doi: 10.1007/BF01530777.

[37] Alon, U. J.: An Introduction to Systems Biology: Design Principles of Biological Circuits. 1st Edition, Chapman and Hall/CRC Mathematical and Computational Biology, Chapman and Hall/CRC, 2006.

[38] Rumelhart, D.—McClelland, J. L.: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations. Cambridge, MA, MIT Press, 1986.

[39] Douglas, R. J.—Martin, K. A. C.: Recurrent Neuronal Circuits in the Neocortex. Current Biology, Vol. 17, 2007, No. 13, pp. 496–500, doi: 10.1016/j.cub.2007.04.024.

[40] Dawkins, R.: The Blind Watchmaker. Longman Scientific and Technical, 1986.

[41] Wrzeszcz, M.—Koźlak, J.—Kitowski, J.: Modelling Agents Cooperation Through Internal Visions of Social Network and Episodic Memory. Computing and Informatics. Vol. 36, 2017, No. 1, 86–112, doi: 10.4149/cai_2017_1_86.

[42] Fernández-Domingos, E.—Loureiro, M.—Álvarez-López, T.—Burguillo, J. C.—Covelo, J.—Peleteiro, A.—Byrski, A.: Emerging Cooperation in N-

Person Iterated Prisoner's Dilemma over Dynamic Complex Networks. Computing and Informatics, Vol. 36, 2017, No. 3, pp. 493–516, doi: 10.4149/cai_2017_3_493.

[43] Huang, G.-B.: Learning Capability and Storage Capacity of Two-Hidden-Layer Feedforward Networks. IEEE Transactions on Neural Networks, Vol. 14, 2003, No. 2, pp. 274–281.

[44] Ojha, V. K.—Dutta, P.—Chaudhuri, A.—Saha, H.: Understating Continuous Ant Colony Optimization for Neural Network Training: A Case Study on Intelligent Sensing of Manhole Gas Components. International Journal of Hybrid Intelligent Systems, Vol. 12, 2015, No. 4, pp. 185–202.

[45] Abd Elrahman, S. M.—Abraham, A.: Class Imbalance Problem Using a Hybrid Ensemble Approach. International Journal of Hybrid Intelligent Systems, Vol. 12, 2015, No. 4, pp. 219–227.

[46] dos Santos, G. A. M.—Ferrão, V. T.—Vinhal, C. D. N.—da Cruz, G.: Performance Analysis for a Novel Adaptive Algorithm for Realtime Point Cloud Ground Segmentation. International Journal of Hybrid Intelligent Systems, Vol. 12, 2015, No. 4, pp. 229–243.

[47] Samuel, O. W.—Asogbon, G. M.—Sangaiah, A. K.—Fang, P.—Li, G.: An Integrated Decision Support System Based on ANN and Fuzzy AHP for Heart Failure Risk Prediction. Expert Systems with Applications, Vol. 68, 2017, pp. 163–172, doi: 10.1016/j.eswa.2016.10.020.

[48] Salcedo-Sanz, S.: Modern Meta-Heuristics Based on Nonlinear Physics Processes: A Review of Models and Design Procedures. Physics Reports, Vol. 655, 2016, pp. 1–70, doi: 10.1016/j.physrep.2016.08.001.

[49] Yu, K.—Li, Y.—Cai, Z.: A Hybrid Generic Algorithm for Dynamic Data Mining in Investment Decision Making. International Journal on Data Science and Technology, Vol. 2, 2016, No. 6, pp. 62–71.

[50] Vivekanandan, T.—Sriman Narayana Iyengar, N. Ch.: Optimal Feature Selection Using a Modified Differential Evolution Algorithm and Its Effectiveness for Prediction of Heart Disease. Computers in Biology and Medicine, Vol. 90, 2017, pp. 125–136, doi: 10.1016/j.compbiomed.2017.09.011.

Lidio Mauro Lima de Campos is Adjunct Professor at the Faculty of Computing at the Federal University of Pará. He has his B.Sc. in electrical engineering and informatics from this same university, his M.Sc. degree in computer science from the Federal University of Santa Catarina and Ph.D. degree in electrical engineering from the Federal University of Pará. His research interests are in the field of machine learning, neural networks and hybrid intelligent systems.

Roberto Célio Limão de Oliveira graduated in electrical engineering from Universidade Federal do Pará (1987), received his Master's in electrical engineering from Instituto Tecnológico de Aeronáutica (1991) and Ph.D. in electrical engineering from Universidade Federal de Santa Catarina (1999). He has experience in electrical engineering, focusing on computational intelligence, artificial neural networks and evolutionary computing.

Gustavo Augusto Lima de Campos is Associate Professor at UECE with an expertise in the field of artificial intelligence, mainly intelligent agents to solve decision problems in complex environment, based on systematic and local search methods, neural networks, logic and fuzzy systems. He obtained his Doctor degree in electrical engineering at the Systems Engineering Department at the Electrical and Computing Engineering Faculty of the State University of Campinas, São Paulo, in 2003.