Expert Systems With Applications 56 (2016) 368–384

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Optimization of neural networks through grammatical evolution and a genetic algorithm

Lídio Mauro Lima de Campos[a,*], Roberto Célio Limão de Oliveira[a], Mauro Roisenberg[b]

[a] Faculty of Computer Engineering, Federal University of Pará, Rua Augusto Corrêa 01, Guamá, CEP:66075-100, Belém Caixa postal 479, Pará Brazil
[b] Informatic and Statistic Department, Federal University of Santa Catarina, PO. Box 476, Trindade, Florianópolis 88040-900, Santa Catarina, Brazil

A R T I C L E   I N F O

Article history:
Received 17 August 2015
Revised 18 January 2016
Accepted 8 March 2016
Available online 16 March 2016

Keywords:
Evolutionary computation
Neural networks
Grammatical evolution

A B S T R A C T

This paper proposes a hybrid neuro-evolutive algorithm (NEA) that uses a compact indirect encoding scheme (IES) for representing its genotypes (a set of ten production rules of a Lindenmayer System with memory), moreover has the ability to reuse the genotypes and automatically build modular, hierarchical and recurrent neural networks. A genetic algorithm (GA) evolves a Lindenmayer System (L-System) that is used to design the neural network's architecture. This basic neural codification confers scalability and search space reduction in relation to other methods. Furthermore, the system uses a parallel genome scan engine that increases both the implicit parallelism and convergence of the GA. The fitness function of the NEA rewards economical artificial neural networks (ANNs) that are easily implemented. The NEA was tested on five real-world classification datasets and three well-known datasets for time series forecasting (TSF). The results are statistically compared against established state-of-the-art algorithms and various forecasting methods (ADANN, ARIMA, UCM, and Forecast Pro). In most cases, our NEA outperformed the other methods, delivering the most accurate classification and time series forecasting with the least computational effort. 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 and simulating dynamical systems.

1. Introduction

Artificial Neural Networks (ANNs) are applied in classification (Rivero, Dorado, Rabual, & Pazos, 2010), control systems (Li, Jia, Liu, & Ding, 2014), prediction (Donate, Sanchez, & de Miguel, 2012), and many other problems (Czajkowski, Patan, & Szymański, 2014). As the network architecture depends on the class of problem to be solved (classification, time series forecasting, or reinforcement learning), the full potential of an ANN can be exploited by optimizing its architecture and training. The architecture is designed by heuristic trial-and-error methods that select the transfer function and training algorithm for adjusting the synaptic weights. These heuristics affect the learning and generalization capacities of the network and are usually performed by experts, who sequentially train different topologies to determine the best architecture for a specific problem. To mitigate these deficiencies, researchers have proposed automatic procedures for these processes.

Neuro-evolutive algorithms (NEAs) design and/or train ANNs through evolutionary algorithms (EAs). Bio-inspired algorithms (BIOAs) have gained popularity as efficient solvers of nonlinear optimization problems (Krömer, Platoš, & Snášel, 2014). In this paper, we investigate whether a new biologically inspired NEA can provide an efficient automatic design tool for ANNs.

As discussed by (Stanley & Miikkulainen, 2003) , the increasing complexity of evolutionary computation demands more sophisticated methods than direct mapping from genotype to phenotype , (Dasgupta & McGregor, 1992; Donate et al., 2012; Miller, Todd, & Hegde, 1989). An indirect encoding system (IES) allows a more compact and scalable representation than a direct encoding scheme (DES) (Ahmadizar, Soltanian, & AkhlaghianTab, 2015; Hornby & Pollack, 2002; Lee, Seo, & Sim, 2007; Soltanian, Tab, Zar, & Tsoulos, 2013; Stanley, D'Ambrosio, & Gauci, 2009).

Building an efficient encoding scheme for repetitive and recurrent structures is a challenging task for IES-based NEAs. The studies of (Ahmadizar et al., 2015; Dasgupta & McGregor, 1992; Donate et al., 2012; Lee et al., 2007; Miller et al., 1989; Niska, Hiltunen, Karppinen, Ruuskanen, & Kolehmainen, 2004; Sanchez & Melin, 2014; Soltanian et al., 2013; Tsoulos, Gavrilis, & Glavas, 2008) were limited to multilayer perceptron neural

* Corresponding author. Tel.: +55 91 3201 7901.
E-mail addresses: lidiomauro@hotmail.com, lidio@ufpa.br (L.M.L. de Campos), limao@ufpa.br (R.C.L. de Oliveira), mauro@inf.ufsc.br (M. Roisenberg).

http://dx.doi.org/10.1016/j.eswa.2016.03.012
0957-4174/© 2016 Elsevier Ltd. All rights reserved.

networks (MLPs). However, dynamical systems cannot be precisely estimated by MLPs, and they require recurrent neural networks (RNNs). NEAs evolving RNNs are rarely reported in the literature (Beer & Gallagher, 1992; Hornby & Pollack, 2002). The present paper proposes that the brain manipulates recurrent subnetworks with specialized functions. Such architecture is more similar to biological neural networks than feedforward neural networks and realizes more complex computation.

Our approach 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 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 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).(Hornby & Pollack, 2002; Lee et al., 2007) adopted L-systems. Although Lee et al. (2007), used DNA encoding, their study was restricted to feedforward neural networks, whereas our approach is extended to recurrent networks. In the IES used by Hornby and Pollack (2002), 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 Hornby and Pollack (2002)'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 (Eq. 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. In this sense, our model automates the methodology of (Stanley et al., 2009).

In addition, we investigate whether increasing or decreasing the chromosome length affects the crossover rate, and consequently the number of valid production rules. Our experiments confirm interference between the chromosome length, crossover rate, and production rules. The variable associations are discussed in Section 6.1.

The next section presents state-of-the-art methodologies and the findings of other authors. Section 3 presents the theoretical foundation of our work. The components of ADEANN and system operation are detailed in Section 4. Section 5 describes the materials and methods and Section 6 presents the general experimental design and analyzes the computational results. A general discussion and final conclusions are presented in Sections 7 and 8.

2. State-of-the-art techniques

This section discusses existing research on NEAs. Most NEAs use DESs, which specify every connection and node in the genotype that will appear in the phenotype (ANN) . In the approaches proposed by Dasgupta and McGregor (1992); Miller et al. (1989) a multilayer perceptron is encoded as a directed graph, and its genotype is represented by an adjacency matrix . The ANN architecture is evolved by the ANN itself. In other words, the chromosome encodes all details of the network architecture. The weights of the neural network can then be optimized by gradient-based search approaches. These methods are simply implemented, but the size of the connectivity matrix scales as the square of the number of nodes. Thus, this representation blows up as the number of nodes increases.

Stanley and Miikkulainen (2002) proposed neuro-evolution of augmenting topologies (NEAT). In NEAT, the DES incorporates a few biologically plausible entities, and alters both the weighting parameters and structures of the networks. The flexible genetic encoding of NEAT describes a network by a list of edge genes. Each edge gene denotes an edge between two node genes, and specifies the in-node, out-node, and weight of the edge. The data structure of NEAT, which represents the genotype, grows linearly with the number of edges between two nodes.

Donate et al. (2012) proposed a new approach for automatic ANN design, and applied it to time-series forecasting using a GA. Their method alters both the weighting parameters and network structures. The objective was to improve the accuracy of time-series forecasting. However, they restricted their approach to MLP, which is unsuitable for simulating temporal series forecasting.

Sanchez and Melin (2014) developed a novel multi-objective optimization for a hierarchical genetic algorithm (MOHGA) based on the micro-GA approach. Their method, which optimizes modular neural networks (MNNs), is used in iris recognition. The MOHGA automatically divides the data into granules or submodules, and selects the data for the training and testing phases. It determines the number of granules or submodules and the percentage of the training data to improve the results. The applicability of MOHGA was confirmed in human recognition. The MMNs delivered significantly better learning outcomes than single ANN, especially in backpropagation (BP). However, this method was not tested in other applications such as temporal series forecasting and classification. Moreover, it cannot yield the recurrent neural networks required in dynamical systems simulations.

At the other extreme are IESs (Ahmadizar et al., 2015; Boozarjomehry & Svrcek, 2001; de Campos, Roisenberg, & de Oliveira, 2011; Kitano, 1990; Lee et al., 2007; Risi & Stanley, 2012a; Soltanian et al., 2013; Stanley et al., 2009; Tsoulos et al., 2008). In an IES, the network generation is indirectly specified by the genotype. The solution description is compressed, enabling complex topologies of ANNs. Later, we will discuss several IES-based methods in a so-called generative and developmental system (GDS). GDSs constitute a sub-field of evolutionary computation; namely, IES evolution. In this representation, the phenotype is 'grown' via a developmental process specified by the genotype.

Stanley et al. (2009) to extent NEAT Stanley and Miikkulainen (2002) incorporated IES into a hypercube-based NEAT. Their HyperNEAT generates a neural network by compositional pattern producing networks (CPPNs), and abstracts the development. HyperNEAT describes natural CPPNs patterns as high-level compositions of functions representing several common patterns. (Stanley et al., 2009) HyperNEAT is an effective tool for reinforcement learning. Despite its novel capabilities, the user must manually place the hidden nodes at the substrate, which substantially disadvantages the original HyperNEAT.

The original HyperNEAT was improved by ES-HyperNEAT (Gauci & Stanley, 2010; Risi & Stanley, 2012a). The HyperNEAT CPPN encodes an infinite number of weights within the hypercube, from which a subset should be chosen for incorporation into the ANN. If the added entity is a connection, the connected nodes must also be included. Risi and Stanley (2012a) considered that the density of nodes was also important, but found no additional advantage at higher node densities. They controlled and measured the node density by the quadtree algorithm.

The following methods belong to the class of GDSs using grammatical evolution (GE).

Lee et al. (2007) adopted biologically inspired approaches based on DNA. They encoded information by the real nucleotide symbols A, G, T, and C. A sequence of three of these symbols is known as a codon. The codon sequence between two delimiters is translated into a production rule for a neural controller. Their method is restricted to MLPs, and so cannot generate RNNs.

Hornby and Pollack (2002) defined a class of representations called generative representations, which characteristically reuse elements of the genotype when translating to the phenotype. These authors examined body-brain evolution by associating L-systems and edge encoding. They explored the concurrent morphological evolution and neural controller of simulated robots, and also introduced GENRE, an evolutionary system for evolving designs in this representation.

Using GE, Tsoulos et al. (2008) designed and trained MLPs with one hidden layer. Soltanian et al. (2013) designed a network topology by GE and optimized the weights through BP. Network evaluations against training data were also evaluated by BP. However, their method is also restricted to MLPs. Ahmadizar et al. (2015) proposed the GEGA NEA, which newly combines GE and GA and simultaneously evolves the topology and the connection weights of the MLPs. GE designs the network topology, while GA adapts the weights. Moreover, because over-fitting deteriorates the generalization ability of ANNs, the algorithm simplifies the evolved ANN by a novel adaptive penalty approach. Consequently, GEGA produces greatly simplified ANNs with high generalization ability. However, the applicability of GEGA to other applications, such as temporal series-forecasting, was not tested, and the model cannot yield RNNs.

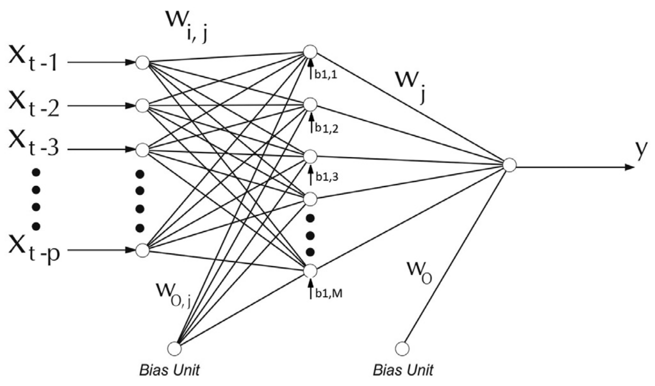


Fig. 1. The first network model generated by ADEANN for classification problems.

State-of-the-art techniques are discussed in detail in de Campos and de Oliveira (2013). The next section presents the theoretical foundation of our work.

3. Theoretical foundation

The Lindenmayer system (L-system) is a language for modeling growth. Let Σ denote an alphabet, Σ^* be the set of all words over Σ and Σ^+ be the set of all nonempty words over Σ. An L-system is an ordered triplet $G = \{\Sigma, \Pi, \alpha\}$ where $\Pi \subset \Sigma \times \Sigma^*$ is a finite set of productions rules and $\alpha \in \Sigma^+$ is a nonempty word called the axiom. A production rule $(a,\chi) \in \Pi$ is written as $a \rightarrow \chi$. The letter a and the word χ are called the predecessor and the successor of the production, respectively. It is assumed that for any letter $a \in \Sigma$, there exists at least one word $\chi \in \Sigma^*$ such that $a \rightarrow \chi$ (Lindenmayer, 1968).

Genetic algorithms (GAs) are computer programs that solve problems and model evolutionary systems by mimicking the processes of biological evolution. In science and engineering, GAs are useful adaptive algorithms for solving practical problems and modeling natural evolutionary systems (Mitchell, 1998).

A neural network consists of a set of interconnected artificial neurons, structured in layers, nodes, perceptrons, or groups of processing units, which process and transmit information through activation functions. The nodes of one layer are connected to, and send information to, the nodes of the next layer. The connections between nodes i and j of the previous and following layers are associated with a weight w_{ij}. Each neuron is also assigned a bias b_i. Fig. 1 illustrates the first network model used in this work. This ANN architecture is known as feedforward. The neural network learning is based on several inputs(x)/output (o) pairs. During the learning process, also known as the training process, the weights of the connections between the nodes of successive layers are adjusted. The second ANN model employed in this work is shown in Fig. 2. This recurrent neural network has a standard MLP architecture, Fig. 1, plus additional loops for temporal processing and sequence learning by the network.

In the next section we present a NEA that incorporates some aspects of bilogical inspiration.

4. Proposed NEA

Our NEA, called ADEANN, links three computational paradigms inspired by natural phenomena; namely, GA, an L-system and an ANN. The GA mimics biological evolution at a high level of abstraction. The evolutionary processes are defined as temporal genetic changes in the hypothetical DNAs of a population of individuals, regulated by a selection mechanism. The hypothetical DNA of

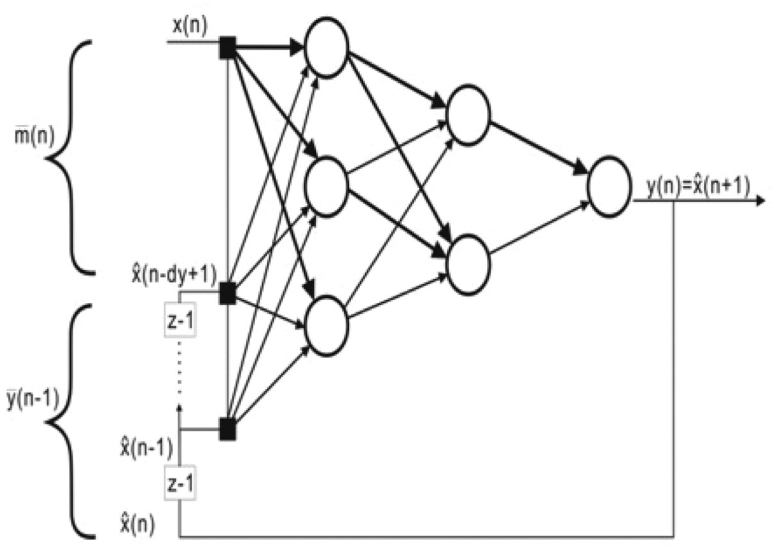


Fig. 2. The second network model generated by ADEANN for the experiments of time series forecasting.

Table 1
Production rules of the parametric L-system with memory.

Rule identifier	Rule
1,2	S → **.** (axiom) (2) . → (f...f)n
3	(3.1)f → [f (3.2)f → fF(w,l)f (3.3)f → fF(w,l) (3.4)f → n
3,4,5	(3.5)f → f (3.6)f → fB (4) [→ [F(w,l)f] (5) f → f∗

each individual encodes a set of production rules of an L-system, which coordinates the growth of artificial neurons. These neurons are grouped into ANNs. Each ANN is trained to solve a given problem (classification or prediction) and is thereafter tested to check its generalization or prediction capability. Based on these criteria, each individual is ranked according to its assigned fitness value. The selection mechanism chooses a subset of individuals to be manipulated by the genetic operators (mutation and crossover), whereby a new population is obtained. These processes are repeated through n generations. Finally, the best ANN among the current population is selected.

The general structure of ADEANN is shown in Fig. 3. The optimization process of ADEANN proceeds through several stages. The GA starts with a population of individuals randomly initialized with 0s and 1s (Fig. 3a). Second, the bits of each individual of the population are subjected to transcription (Fig. 3b) and translation (Fig. 3c), 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 (Fig. 3d). All of the genotypes are mapped to phenotypes (ANN architectures) (Fig. 3e). The ANNs are then trained (Fig. 3f) and validated, and tests are carried out. The classification accuracy of each ANN is measured from its fitness (Fig. 3g), calculated by Eqs. (4) or (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 (Fig. 3h). The GA selects the best individuals (Fig. 3i) for mutation and crossover operations (Fig. 3j), which provide the new population (Fig. 3k). The previous steps are repeated through *n* generations. Finally, the simulation results are generated and processed by statistical analyses.

In the next subsection, we discuss the neural codification process of the L-system.

4.1. Neural codification by the Lindenmayer system

To mimic the mechanism of grown structures, including the neurons, we adopt a parametric L-system with memory (Lindenmayer, 1968). Our L-system 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) are described in Table 1. The axiom α = . 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. The third rule is divided into six sub-rules.

Our approach is biologically motivated by the human brain, which contains many more neurons than nucleotides in the genome (Gruau, 1994).

To paraphrase (Gruau, 1994), 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 $2 * 10^9$. Gruau (1994) 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, Fig. 4 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 use, represents a branch with three neurons (see last row, third column of Fig. 4).

Rule (5), $f \rightarrow f*$ (which means $f \rightarrow f[Ff]Ff]Ff]$, 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 Fig. 4. This phenotype begins a new branch from N3. Here, we have used the principle of *reuse*

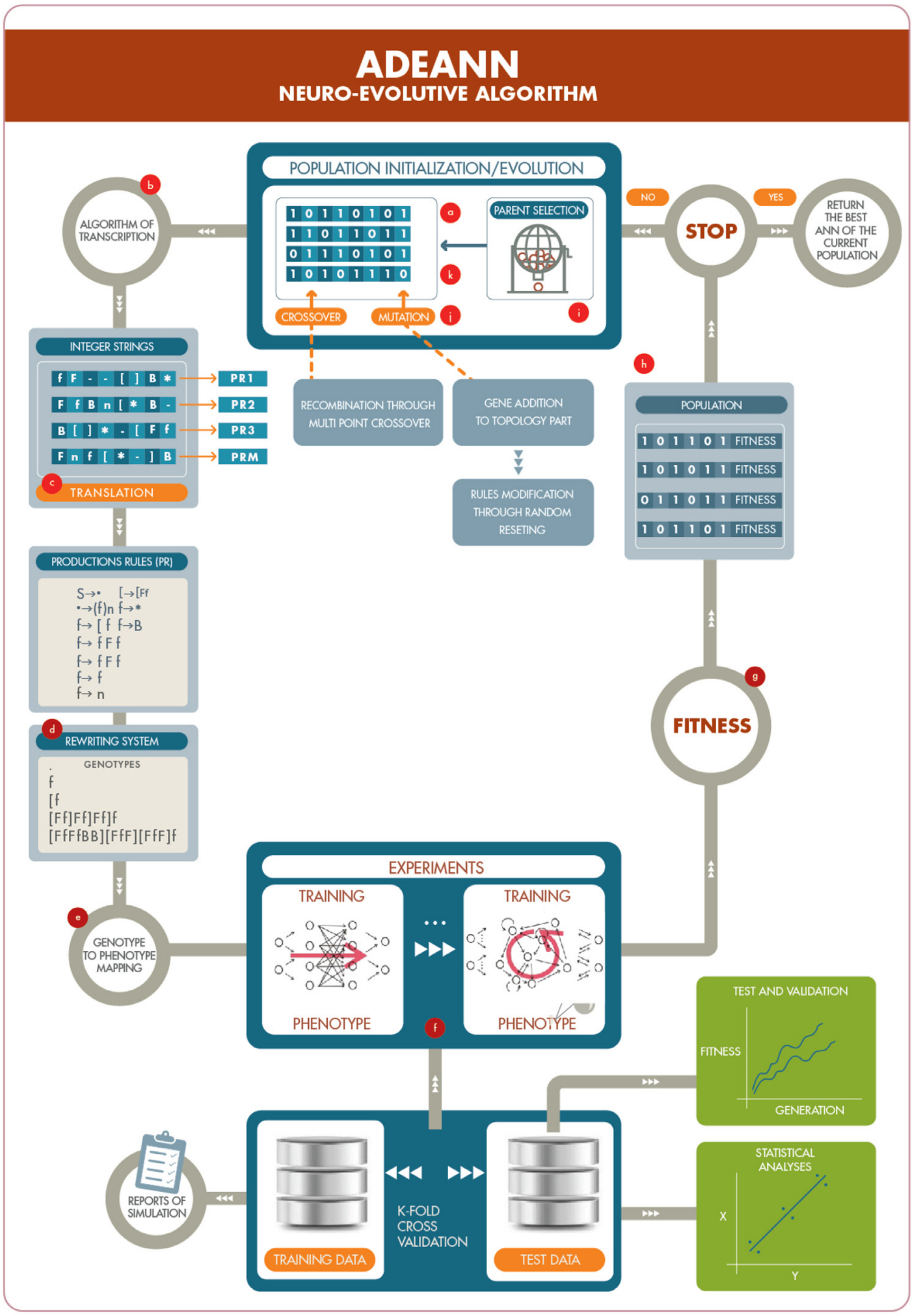


Fig. 3. Decision making process in the proposed approach.

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]f**, with phenotype shown in the third row and third column of Fig. 4. 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]f**, yielding **[Ff[Ff]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[Ff]Ff]Ff]]Ff[FfF]Ff]]Ff]f**. This step creates a synapse from N41 to N8. Fig. 4 also shows the mapping between genotypes and phenotypes. Our indirect encoding method, presented in subsection 3.2, generates ANNs with variable numbers of neurons in the range [*X, Y*], where *X* and *Y* are computed by Eqs. (1) and (2), respectively.

$$X = [Ninputs + Ninputs * random(NRmin) + Noutputs] \quad (1)$$

$$Y = [Ninputs + Ninputs * random(NRmax) + Noutputs] \quad (2)$$

Here, *Ninputs* and *Noutputs* 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, Fig. 5(a) illustrates the iterative generation of ANN at each stage of the development process. The

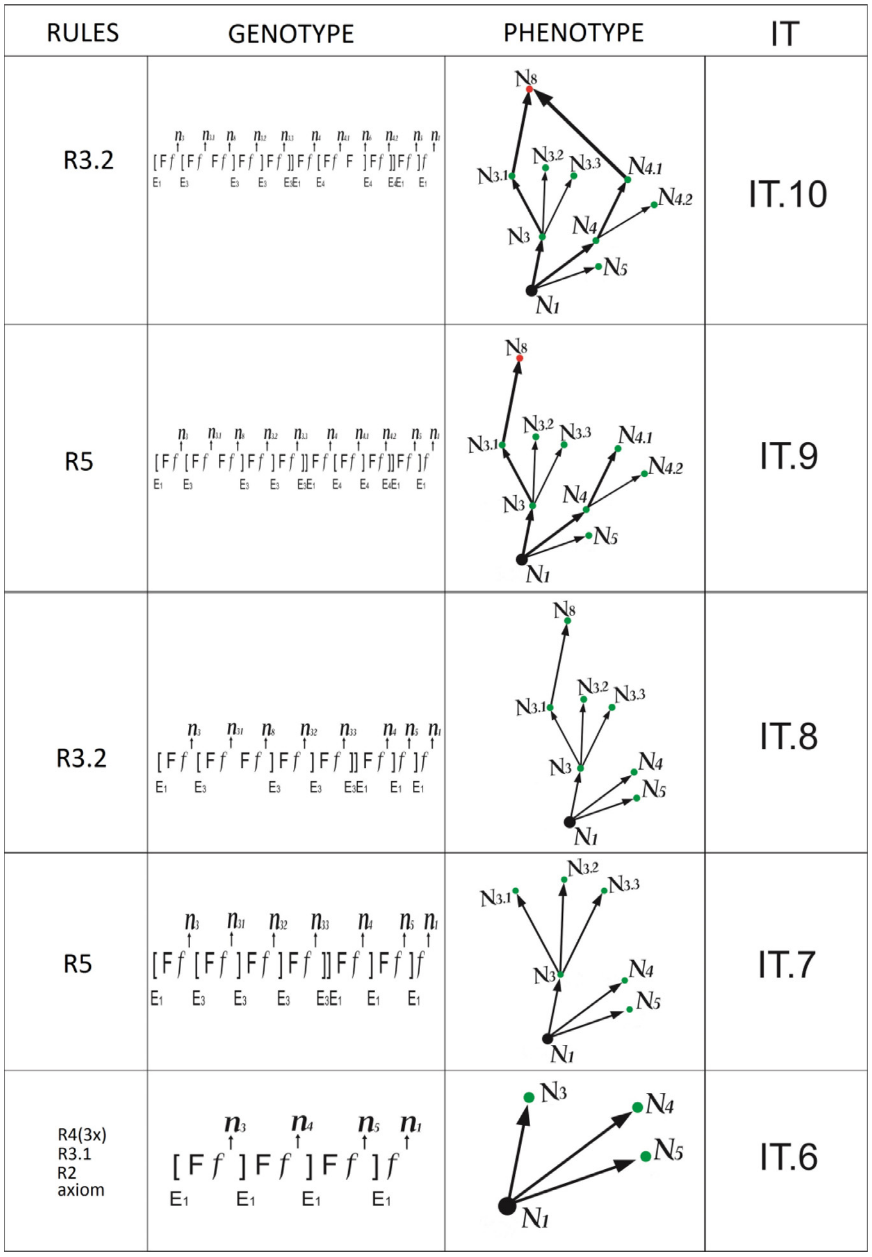


Fig. 4. Construction process of one branch of an iterated ANN (IT denotes the number of iterations).

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 Fig. 5(b). The entire developmental process follows an important biological property called the *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.

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 are described in Section 4.1 and illustrated in Table 1). The recursive rules in Table 1 drive the developmental stages of the neurons (the developmental process is described in Section 4.1 and illustrated in Fig. 4).

Dawkins (1986) 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 (Fig. 6(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, G, 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

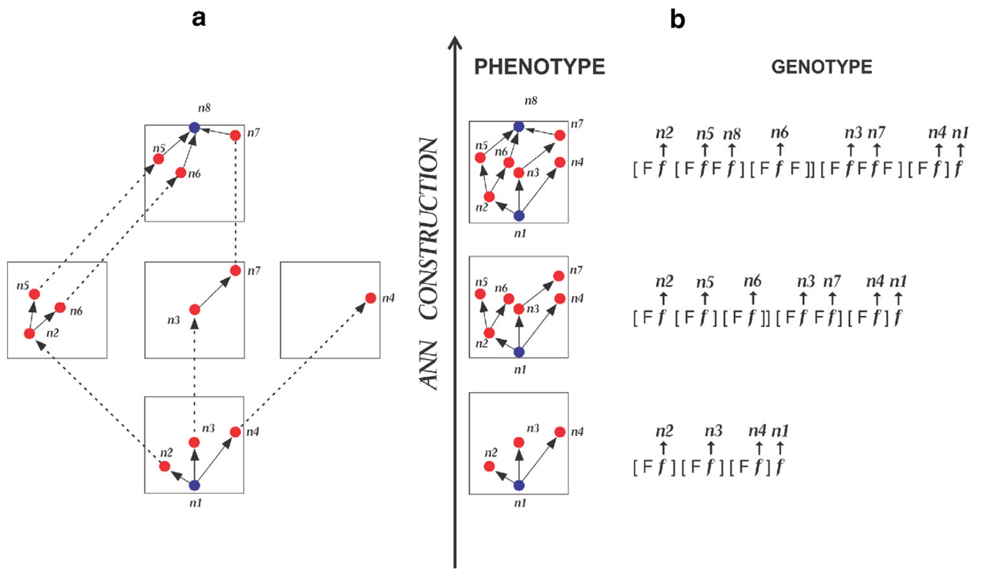
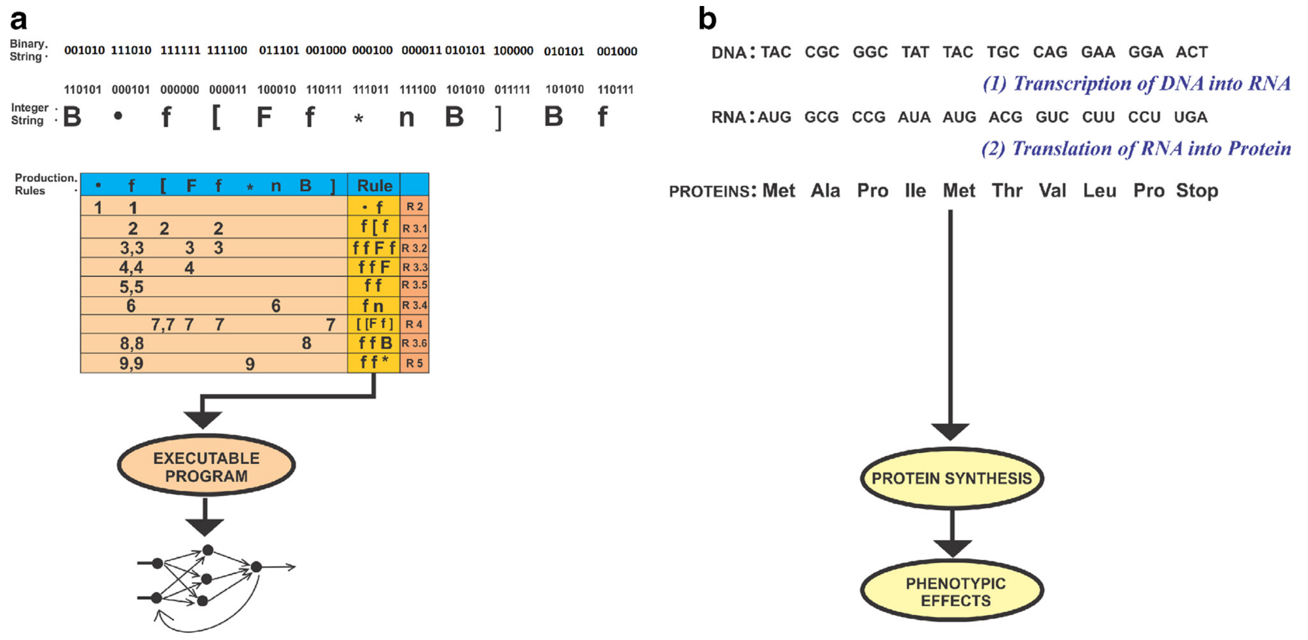

a
b
PHENOTYPE
GENOTYPE
ANN CONSTRUCTION
n1
n2
n3
n4
n5
n6
n7
n8
n2 n5 n8 n6 n3 n7 n4 n1
[F f [F f F f] [F f F]] [F f F f F] [F f] f
n2 n5 n6 n3 n7 n4 n1
[F f [F f] [F f]] [F f F f] [F f] f
n2 n3 n4 n1
[F f] [F f] [F f] f

Fig. 5. Construction of an iterated network.


a
Binary String : 001010 111010 111111 111100 011101 001000 000100 000011 010101 100000 010101 001000
110101 000101 000000 000011 100010 110111 111011 111100 101010 011111 101010 110111
Integer String : B • f [F f * n B] B f
Production Rules :
EXECUTABLE PROGRAM
b
DNA: TAC CGC GGC TAT TAC TGC CAG GAA GGA ACT
(1) Transcription of DNA into RNA
RNA: AUG GCG CCG AUA AUG ACG GUC CUU CCU UGA
(2) Translation of RNA into Protein
PROTEINS: Met Ala Pro Ile Met Thr Val Leu Pro Stop
PROTEIN SYNTHESIS
PHENOTYPIC EFFECTS

•	f	[	F	f	*	n	B	]	Rule	
1	1								• f	R 2
	2	2		2					f [f	R 3.1
	3,3		3	3					f f F f	R 3.2
	4,4		4						f f F	R 3.3
	5,5								f f	R 3.5
	6					6			f n	R 3.4
		7,7	7	7				7	[[F f]	R 4
	8,8						8		f f B	R 3.6
	9,9				9				f f *	R 5

Fig. 6. (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.

leucine (Leu). In Fig. 6(b), the protein is formed by a sequence of amino acids starting with methionine (Met) and ending with proline (Pro). Such protein synthesis triggers all stages of the neuronal development (phenotypic effects), as shown in Fig. 6(b). The information flow DNA → transcription → RNA → translation → protein is referred to as the central dogma in molecular biology.

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 (the message contained in DNA). Each two-bit sequence represents one nucleotide; for example, the set (00, 01, 10, 11) symbolizes (U, C, A, G) in the original genetic code. Accordingly, six bits represent three nucleotides; that is, (000000, 011111) symbolizes (UUU, 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. The metaphoric information flow in the artificial model is the binary string → transcription → Integer String → translation → Production Rules (see Fig. 6(a)).

Steps 1 and 2 of the function Rule-extraction-with-GA mimic the DNA transcription process into RNA, as shown in Fig. 6(b). These steps are repeated for all individuals in the population. All the integer strings obtained after step 4 of this process (see Fig. 6(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'.

Fig. 6(a) illustrates the rule extraction (translation of an integer string into valid production rules) 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

Table 2
The genetic code from the perspective of mRNA, translated as in Fig. 6(b). In the same table, the DNA's metaphor.

	00 (U)	01 (C)	10 (G)	11 (A)	
00 (U)	**f** (UUU) Phe	**F** (UCU) Ser	**n** (UAU) Tyr	**.** (UGU) Cys	**00** (U)
00 (U)	**n** (UUC) Phe	**.** (UCC) Ser	**f** (UAC) Tyr	**F** (UGC) Cys	**01** (C)
00 (U)	**F** (UUA) Leu	**f** (UCA) Ser	**B** (UAA) Stop	**f** (UGA) Stop	**10** (A)
00 (U)	**[** (UUG) Leu	**n** (UCG) Ser	**[** (UAG) Stop	**∗** (UGG) Trp	**11** (G)
01 (C)	**f** (CUU) Leu	**]** (CCU) Pro	**n** (CAU) His	**∗** (CGU) Arg	**00** (U)
01 (C)	**∗** (CUC) Leu	**F** (CCC) Pro	**f** (CAC) His	**F** (CGC) Arg	**01** (C)
01 (C)	**]** (CUA) Leu	**f** (CCA) Pro	**∗** (CAA) Gln	**[** (CGA) Arg	**10** (A)
01 (C)	**f** (CUG) Leu	**∗** (CCG) Pro	**B** (CAG) Gln	**]** (CGG) Arg	**11** (G)
10 (A)	**∗** (AUU) Ile	**]** (ACU) Thr	**n** (AAU) Asn	**f** (AGU) Ser	**00** (U)
10 (A)	**f** (AUC) Ile	**B** (ACC) Thr	**f** (AAC) Asn	**B** (AGC) Ser	**01** (C)
10 (A)	**F** (AUA) Ile	**[** (ACA) Thr	**B** (AAA) Lys	**n** (AGA) Arg	**10** (A)
10 (A)	**∗** (AUG) Met	**f** (ACG) Thr	**∗** (AAG) Lys	**]** (AGG) Arg	**11** (G)
11 (G)	**]** (GUU) Val	**[** (GCU) Ala	**F** (GAU) Asp	**n** (GGU) Gly	**00** (U)
11 (G)	**n** (GUC) Val	**B** (GCC) Ala	**[** (GAC) Asp	**.** (GGC) Gly	**01** (C)
11 (G)	**f** (GUA) Val	**]** (CGA) Ala	**B** (GAA) Glu	**F** (GGA) Gly	**10** (A)
11 (G)	**B** (GUG) Val	**f** (GCG) Ala	**∗** (GAG) Glu	**[** (GGG) Gly	**11** (G)

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 Fig. 6(a)). For example, Rule 2 (. → f), symbolically represented by (.f), is found at positions 1 and 2 of the string **.f[Ff ∗nB]**. Rule 3.1 f → [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 Fig. 4. The resulting population of ANNs is trained and the fitness of each ANN is calculated by Eq. (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 (Fig. 6(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.

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 subsection 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 a integer string as illustrated in Fig. 6(a).

Table 3
Relation between tournament size and selection intensity.

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

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.

4.3. Fitness evaluation

The fitness of an ANN can be appropriately measured by the mean square error (MSE) given by Eq. (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 Eq. (3), q is the number of test samples

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

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 Eqs. (4) and (5) below:

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

A1 and A2 are two parameters whose values depend on the problem and the precision requirement.

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

where NNHL is the number of neurons in the hidden layer.

4.4. Selection mechanism

In this paper, the reproducing individuals are selected by tournament selection, one of the most widely used selection strategies in GAs (Miller & Goldberg, 1996). 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 Eq. (7), Tournament selection is expected to improve the fitness of each subsequent generation (Oladele & Sadiku, 2013).

To calculate the selection intensity, we must evaluate Eq. (6). The approximate solution is given by 7, where x and y are the fitness values of the population and t is the tournament size. Table 3 shows how the selection intensity calculated by Eq. (7) varies with tournament size.

In tournament selection, larger tournament sizes t is lead to higher expected loss of diversity (LD) (Oladele & Sadiku, 2013). As evident from Eq. (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. 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.

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

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

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

4.5. Crossover and mutation operators

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 (Kenneth & Spears, 1992). 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 spaces, 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).

4.6. 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 (Fig. 1), is commonly employed in systems biology studies (Alon, 2006), including evolutionary studies. Information flows through the ANN in discrete time. The output o_j of node j is calculated by Eq. (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 (Eq. 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 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 Eq. (11) is a momentum term, introduced by Rumelhart and McClelland (1986) to accelerate the learning process while avoiding instability in the algorithm.

Table 4
Parameters used in the experiments.

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

Table 5
Summary of the datasets used in this paper and our previous paper (de Campos et al., 2015).

Dataset	No.of inputs	No.of classes	Patterns
Breast cancer detection	9	2	699
Iris flower	4	3	150
Heart disease	13	4	351
KDDCUP99	42	5	125,973
Breast cancer prediction	15	3	390

Table 6
Time series descriptions for the training and test sets; In-SP, Out-SP, and K denote In-Samples, Out-Samples, and period, respectively.

Dataset	In-SP	Out-SP	(K)	Normalization
Passengers	122	22	12	Z-score
Temperature	214	25	12	Decimal scaling
Down Jones	241	50	12	Decimal scaling

The second network model generated by ADEANN, which was employed in the time-series forecasting experiments, was the single hidden-layer recurrent network shown in Fig. 2. (Douglas & Martin, 2007) 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.

The next section presents the material and methods of the study.

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

5. Materials and methods

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 7. 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.

5.2. Dataset description

To validate the ADEANN and compare it with other NEAs, we selected four real-world datasets used in classification tasks from the UC Irvine Machine Learning Repository at the University of California (Blake & Merz, 1998). These datasets, summarized in Table 5, were the Breast Cancer Detection, Iris Flower and Heart

Table 7
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.

GxP	ANN	CL	(c,m)	Cr	μ	σ	MSE	PR %
E1: 50 × 30	**(2,8,1)**	**180**	**(9,3)**	**0.6**	**6.50**	**1.50**	**0.000046**	6.67
E2: 50 × 30	(2,7,1)	216	(10,3)	0.6	4.80	2.32	0.000049	16.67
E3: 50 × 30	(2,5,1)	252	(12,4)	0.6	4.80	1.72	0.000048	25
E4: 60 × 30	(2,6,1)	252	(12,4)	0.9	3.86	1.81	0.000032	23.33
E5: 50 × 20	(2,6,1)	276	(13,5)	0.6	6.00	1.94	0.000029	20.28
E6: 50 × 30	(2,2,1)	180	(18,3)	0.3	5.67	2.62	0.000076	9
E7: 50 × 30	(2,2,1)	180	(18,3)	0.6	3.20	0.75	0.000075	13.40
E8: 50 × 30	(2,2,1)	180	(18,3)	0.9	4.50	2.18	0.000051	11.33
E9: 50 × 20	(2,2,1)	216	(32,28)	0.6	6.00	2.07	0.000056	22.33
E10: 50 × 20	(2,2,1)	216	(32,28)	0.9	3.71	2.43	0.000056	21.53
E11: 50 × 30	(2,2,1)	252	(37,33)	0.6	3.89	3.65	0.000056	28.67
E12: 60 × 30	(2,2,1)	252	(37,33)	0.9	4.36	1.72	0.000055	26.73
E13: 50 × 30	(2,2,1)	300	(45,40)	0.6	3.31	1.65	0.000053	51.20
E14: 60 × 30	(2,2,1)	300	(45,40)	0.9	3.77	2.01	0.000053	41.93
E15: 50 × 30	(2,2,1)	360	(54,48)	0.6	3.69	2.14	0.000056	50.73
E16: 50 × 30	(2,2,1)	360	(54,48)	0.9	3.71	1.87	0.000056	54.33
E17: 50 × 30	(2,2,1)	390	(58,52)	0.6	3.79	1.61	0.000053	63.33
E18: 50 × 30	(2,2,1)	390	(58,52)	0.9	5.10	2.05	0.000053	63.40
E19: 100 × 30	**(2,2,1)**	516	**(77,68)**	**0.9**	**5.28**	**2.03**	**0.000024**	96.67
E20: 100 × 30	(2,2,1)	516	(77,68)	0.6	4.15	2.17	0.000025	90

Table 8
Comparison between ADEANN and other methods in the XOR problem.

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

[a] Average classification accuracy.
[b] Average number of hidden neurons.
[c] Standard deviation of the number of hidden neurons.
[d] MSE, DENN (Ilonen et al., 2003).

Disease datasets, and the KDDCUP99 and Breast Cancer Prediction dataset used in our previous study (de Campos, de Oliveira, & Roisenberg, 2015). For time-series forecasting, we selected the three well-known datasets Passengers, Temperature, and Dow–Jones. These datasets were used in the NN3 (2007) forecasting competition, which aimed to develop a unique computational intelligence method for time-series forecasting. The Passengers time series gives the number of passengers (in thousands) traveling on an international airline, measured monthly from January 1949 to December 1960. The data were sourced from (Box & Jenkins, 1976). The Temperature time series shows the mean monthly air temperature measured at Nottingham Castle from 1920 to 1939, sourced from (Anderson, 1976). The Dow–Jones series gives the monthly closings of the Dow–Jones Industrial Average from August 1968 to August 1981, sourced from Hipel and McLeod (1994).

5.3. Statistics

We now describe the statistics used to evaluate the classification performance of ADEANN. Initially, the NEA was run several times for each dataset. Performance comparisons with other algorithms were then made by appropriate statistical tests. Similar to (Ahmadizar et al., 2015), we measure the accuracy of our algorithm by *k*-fold cross-validation. In *k*-fold cross-validation, the data are randomly divided into *k* non-overlapping sets D_1, D_2, ..., D_k. , and the algorithm runs *k* times. In each iteration *i*, where ($1 \le i \le k$), , the algorithm is trained with $D \backslash D_i$ sets and tested on set D_i (Ahmadizar et al., 2015). Each experiment was repeated 5 times, giving ($5 \times 2 \times k$ cross validation) runs of the algorithm. Because the runs were independent with randomly generated initial seeds, the independence of the events was assured.

In addition, the significances of the results obtained by ADEANN and the competing NEAs *t*-were evaluated at the 95% confidence level (i.e., *p*-value below 0.05). Similar to Castellani (2013) and Ahmadizar et al. (2015), we statistically compared the algorithm performances by three criteria. The primary (CC1), second (CC2), and third (CC3) criteria were the average classification accuracy, the computational effort, and the average number of hidden neurons, respectively). Similarly, Castellani (2013) and Ahmadizar et al. (2015), compared pairs of algorithms. When there were no significant statistical differences between the CC1, CC2, and CC3 values of two NEAs on a given dataset, they considered that both algorithms performed equally well. In this case, both algorithms are awarded 1 point. On the other hand, if two algorithms obtain significantly different CC1 or CC2 or CC3 scores, the better performing algorithm is awarded 2 points and the other zero points. Consequently, we reject the null hypothesis H_0 and accept the alternative hypothesis H_1 . In deciding whether two performances differ, we test the significance of the difference between u_1 and u_2 ($p < 0.05$). The overall performance of each NEA is then calculated by summing all points achieved in the pairwise comparisons of all datasets. The results are shown in Tables 12, 13 and 14.

Furthermore, we compared the forecasting ability of the ADEANN by autoregressive integrated moving average (ARIMA) (Box & Jenkins, 1976), ADANN (Donate et al., 2012), the unobserved components model (UCM), and a software tool called "Forecast Pro". The characteristics of each time series are summarized in Table 6.

To evaluate the forecast accuracy of ADEANN as well as to compare it among these different methods, we have used two performance measures: MSE and the symmetric mean absolute percentage error (SMAPE) given by Eq. (12). To statistically compare the results of ADEANN and other methods: we chose MSE as the primary criterion (CF1) and SMAPE as the second criterion (CF2). The evaluation method was that used in the classification problems. First, if there is no significant statistical difference between the CF1 and CF2 scores of two NEAs, both algorithms are awarded 1 point. Otherwise, if one method outperforms the other in CF1 or CF2, the higher performing algorithm is awarded 2 points while the lower performing one scores zero points. Consequently, we reject the null hypothesis H_0 and accept the alternative hypothesis H_1. Again, we test whether the difference between u_1 and u_2 is statistically significant ($p < 0.05$). The overall performance of each NEA is then calculated by summing all the points achieved in the pairwise comparisons of all datasets. The results are shown in Table 16.

$$SMAPE = \frac{1}{H} \sum_{t=T+1}^{T+h} \frac{|y_t - F_t|}{(|y_t| + |F_t|)/2} .100\% \tag{12}$$

where T is the number of past values, h denotes the forecasting horizon, y_t is the forecast value, and F_t is the actual value.

The next section presents the simulation results.

6. Results

6.1. Performance analysis of ADEANN components

The results of the XOR problem after twenty experiments are summarized in Table 7. In the last fifteen simulations, the fitness given by Eq. (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

Table 9
Comparison between ADEANN and other methods on the breast cancer dataset.

Dataset	GEGA	ADEANN	G3PCX	GA
B.Cancer (I)	96.61 **3.8**	**99.01**%[a] 8[b]	98.94% 5	98.88% 5
	1 0.05 9200	**0**[c] **0.0003**[d] 25,550[e]	– – 11500	– – **8400**
	MM	ADEANN	GRM	GE-BP
B.Cancer (II)	96.77% **–**	**98.52**%[a] 8[b]	96.71% 5	95.88% **4.6**
	– – 115,000	**0**[c] **0.0004**[d] **33,247**[e]	– – 300,000	0.8 – 92,000
	GEGA	ADEANN	GE	GP
B.Cancer (III)	96.19% **2**	**98.03**%[a] 8[b]	96.02% 3	96.27% 3.3
	0 – 92,000	**0**[c] **0.0003**[d] **75,046**	0.5 – 92000	0.5 – 92,000

[a] Average classification accuracy.
[b] Average number of hidden neurons.
[c] Standard deviation of the number of hidden neurons.
[d] MSE in the generalization phase.
[e] Computational effort.

Table 10
Comparison between ADEANN and other methods on the iris flower dataset.

Dataset	GEGA	ADEANN	G3PCX	GA
Iris (I)	**96.13**% **2.7**	94.11%[a] 4[b]	89.73% 5	88.67% 5
	1.1 – 10,250	**0**[c] **0.001**[d] **6255**[e]	**0** – 10250	**0** – 8200
	MM	ADEANN	GRM	GE-BP
Iris (II)	92.40% **–**	94.11%[a] **4.06**[b]	92.93% –	**95.72**% 4.5
	– – 400,000	**0.24**[c] **0.001**[d] **37124**[e]	– – 1200,000	0.7 – 250,000
	GEGA	ADEANN	GE	GP
Iris (III)	**95.70**% **2.1**	95.10%[a] 4.81[b]	94.93% 3.5	95.22% 8.9
	0.3 – 250,000	1.39[c] **0.001**[d] **225,113**	0.6 – 250,000	0.5 – 320,000

[a] Average classification accuracy.
[b] Average number of hidden neurons.
[c] Standard deviation of the number of hidden neurons.
[d] MSE in the generalization phase.
[e] Computational effort.

Table 11
Comparison between ADEANN and other methods on the heart disease dataset.

Dataset	GEGA	ADEANN	G3PCX	GA
Heart C.(I)	79.55% **2.9**	87.33%[a] 12[b]	**90.42%** 5	87.72% 5
	1.3 – 15,200	**0**[c] **0.001**[d] **10831**[e]	**0** – 19,000	**0** – 15,200
	MM	ADEANN	GRM	GE-BP
Heart C.(II)	76.78% **–**	**87.33%**[a] 12[b]	72.8% –	80.80% **2.5**
	– – 380,000	**0**[c] **0.001**[d] **83810**[e]	– – 600,000	0.7 – 200,000
	GEGA	ADEANN	GE	GP
Heart C. (III)	82.24% **2.4**	**89.33%**[a] 12[b]	79.60% 3.3	80.71% 2.9
	1.0 – 200,000	**0**[c] **0.001**[d] **145,717**	1.4 – 200,000	0.3 – 200,000

[a] Average classification accuracy.
[b] Average number of hidden neurons.
[c] Standard deviation of the number of hidden neurons.
[d] MSE in the generalization phase.
[e] Computational effort.

guaranteed by the fitness function Eq. (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 7, 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 6.

From Table 8, we observe that in 100 runs, NEAT (Stanley, 2002) 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 7) yielded the closest solution, with $\mu = 3.31$ and $\sigma = 1.65$ in 50 generations. ADEANN has a similar average classification accuracy to DENN (Ilonen, Kamarainen, & Lampinen, 2003), 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 30100.

The crossover rate ($Cr = 0.6$ and $Cr = 0.9$ for chromosomes with fewer or more than 300 bits, respectively) increases the number of valid production rules in the population. In all experiments, the mutation rate C_m was set to 0.01. The selected crossover and mutation rates are consistent with the literature. (Lin, Lee, & Hong, 2003) mentioned that typical ranges of Cr and p_m are 0.5–1.0 and 0.001–0.05, respectively. (Ahmadizar et al., 2015) reported

Table 12
Statistical comparison between ADEANN and ANN training algorithms.

Dataset	GEGA	ADEANN	G3PCX	GA
B.cancer (I)	10	6	8	10
Iris(I)	12	14	2	4
Heart ceveland (I)	8	8	8	8
Overall performance (I)	30	28	18	22

Table 13
Statistical comparison between ADEANN and ANN design algorithms.

	MM	ADEANN	GRM	GE-BP
B.cancer (II)	6	12	2	8
Iris (II)	2	14	2	8
Heart cleveland (II)	6	10	4	8
Overall performance (II)	14	36	8	24

Table 14
Statistical comparison between ADEANN and TWEANN algorithms.

	GEGA	ADEANN	GE	GP
B.cancer (III)	8	12	4	6
Iris (III)	14	10	6	4
Heart cleveland (III)	10	12	6	6
Overall performance (III)	32	34	14	16

that high mutation leads to 'noise' optimization behavior and decreases the performance of the algorithm. Larger chromosomes require more crossover and mutation points.

6.2. Comparison with other NEAs in classification problems

In this subsection, the performance of ADEANN is experimentally evaluated and compared with those of existing NEAs. The average classification accuracy of the generated ANNs on test sets is determined by k-fold cross-validation. The average and standard deviation of the number of hidden neurons were measured, along with the computational effort.

Ahmadizar et al. (2015) stated that the CPU time needed to solve a problem cannot properly compare the results of various methods, because it depends on the researchers' programming skills, computer platform, and other factors. Thus, we have used computational effort to measure computational time. Rivero et al. (2010), calculated the computational effort of each algorithm by multiplying the effort of evaluating each individual, the population size and the number of generations. To obtain the effort of evaluating an individual, they counted the number of passes of the training patterns through the corresponding ANN.

The methods are compared by the first, second, and third categories of the NEA. The first category (I) constitute the ANN training algorithms, which evolve only the connection weights. Generalized generation gap parent-centric recombination (G3PCX) (Deb, Anand, & Joshi, 2002) and GA (Paz & Kamath, 2005) are two prominent methods belonging to this class of algorithm. The second category (II) consists of the ANN designing algorithms, which evolve only the topology. Here, the connection weights are evolved by a gradient descent method. Common algorithms for this purpose are the matrix method (MM), the pruning method (PRM), and graph rewriting grammar (GRG), proposed by Paz and Kamath (2005), and GE-BP (Soltanian et al., 2013) are prominent methods. The MM and PRM represent the connectivity of an ANN in a binary matrix, whereas GRG evolves the grammar of the ANN. These algorithms are detailed in Paz and Kamath (2005). The third category (III) simultaneously evolves the topology and the connection weights. Prominent methods in this NEA category are GEGA (Ahmadizar et al., 2015), GP (Rivero et al., 2010) and GE (Tsoulos et al., 2008).

6.2.1. Breast cancer dataset

As the first problem, the ANNs must solve the classification of breast cancer (BC). The dataset (Blake & Merz, 1998) has 699 samples, each with nine features, and each classified as malignant or benign. Each feature is assigned a discrete integer score, from 0 to 10. There are 444 benign samples and 239 malignant samples. The BC dataset was chosen because of the inherent difficulty of breast cancer diagnosis.

All experiments were performed with k=5-fold cross-validation, and each experiment was executed 5 times, giving 25 runs of the algorithm. Table 9 compares the results of ADEANN and G3PCX (Deb et al., 2002) and GA (Paz & Kamath, 2005) taken from (Ahmadizar et al., 2015). In the three most successful experiments (highlighted in boldface), ADEANN finds an ANN for the BC problem in 50 generations. The computational efforts are based on the corresponding results reported in Ahmadizar et al. (2015). The MSEs listed in Table 9 were obtained in the generalization phase.

The results in Table 9 are divided into the three categories of algorithms (I), (II) and (III) discussed in Section 6.2. In the first NEA category (I), the least and most computational effort is expended by GA (Paz & Kamath, 2005) and ADEANN, respectively. However, in all categories, ADEANN yields higher classification accuracy than the other methods, and requires less computational effort than most of the methods in categories (II) and (III). In the first NEA category (I), ADEANN has a lower MSE than GEGA (Ahmadizar et al., 2015) in the generalization phase.

6.2.2. Iris flower dataset

The second problem to be solved is the classification of iris flowers (IF). The flower characteristics are described by four continuous parameters; namely, the longitudes and widths of the petals and the sepals (all measured in mm). The 150 flowers in the dataset belong to three distinct iris species: Setosa, Versicolor and Virginica. All experiments were performed with k=2-fold cross-validation. Each experiment was executed 5 times, giving 10 runs of the algorithm. Table 10 compares the results of ADEANN and the other algorithms; the latter are taken from Ahmadizar et al. (2015). In the best and second-best experiments, ADEANN finds an ANN for the IF problem in 50 generations; in the third-best experiment, it requires 100 generations.

In all NEA categories (I), (II) and (III), the ADEANN requires less computational effort than the other methods. In categories (I) and (III), GEGA achieves the highest classification accuracy, whereas in category (II), GE-GP (Soltanian et al., 2013) obtains the best classification accuracy. In categories (I) and (III), GEGA (Ahmadizar et al., 2015) typically generates the smallest neural networks, but in category (II), ADEANN generates the smallest neural networks.

Fig. 7(c) and (d) plot the average MSE and number of neurons in the hidden layer throughout the evolution of the IF networks. Initially, ADEANN obtains ANNs with a high MSE. The best topologies in subsequent generations were explored by the fitness function Eq. 5. We find that the average MSE and average hidden number of hidden neurons of the population decrease after a few generations, then remain within acceptable limits.

6.2.3. Heart disease dataset

In the third problem, the algorithms must detect the absence or presence of heart disease. These data comprise 13 measurements taken from 303 patients at the Hospital of Cleveland. All experiments were performed with k=5-fold cross-validation. Each experiment was executed 5 times, giving 25 runs of the algorithm.

Table 15
Results of different forecasting methods. CE denotes Computational Effort (a) and NNHL is the average number of hidden neurons (b). - means that no results were reported by the method. TS1, TS2 and TS3 denote the results for Passengers, Temperature and Dow–Jones, respectively.

TS		ADANN	ARIMA	UCM	Forecast Pro	ADEANN
TS1	SMAPE(%)	3.05	3.14	2.37	4.5	**2.30**
	MSE	0.59×10^{-3}	0.41×10^{-3}	$\mathbf{0.28} \times 10^{-3}$	0.75×10^{-3}	0.39×10^{-3}
	CE NNHL	–	—	–	–	359348.0[a] 2[b]
TS2	SMAPE(%)	3.79	3.98	3.38	3.42	**2.07**
	MSE	0.31×10^{-2}	0.29×10^{-2}	0.24×10^{-2}	0.25×10^{-2}	$\mathbf{0.17} \times 10^{-2}$
	CE NNHL	–	–	–	–	445622.40 **2**
TS3	SMAPE(%)	4.76	4.78	4.78	4.78	**0.71**
	MSE	0.10×10^{-1}	0.12×10^{-1}	0.12×10^{-1}	0.12×10^{-1}	$\mathbf{0.26} \times 10^{-4}$
	CE NNHL	–	–	–	–	149919.79 **3**

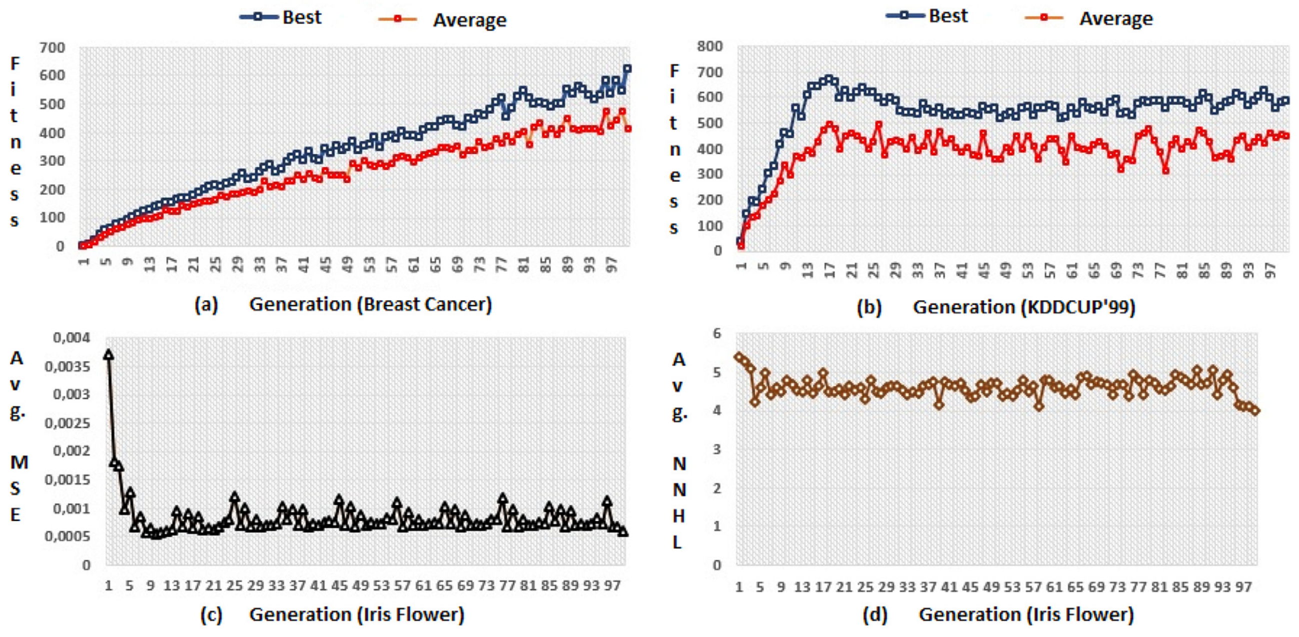


Fig. 7. Results of Simulation - ADEANN's Performance.

Table 16
Statistical comparison between ADEANN and other algorithms (t-test; $p < 0.05$). TS1, TS2 and TS3 represent the results of the Passengers, -Temperature and Dow–Jones time series, respectively, and OP denotes Overall Performance.

TS		ADANN	ARIMA	UCM	Forecast Pro	ADEANN
TS1	SMAPE(%)	4	2	6	0	8
	MSE	2	2	8	2	6
TS2	SMAPE(%)	2	0	6	4	8
	MSE	0	2	4	4	8
TS3	SMAPE(%)	6	0	0	0	8
	MSE	6	0	0	0	8
OP		20	6	24	10	**46**

Table 11 compares the results of ADEANN and the TWEANN algorithms (methods that evolve both the topology and connection weights); the latter are taken from Ahmadizar et al. (2015). In the best experiment, ADEANN finds an ANN for the heart disease problem in 50 generations; in the second and third-best experiments, it requires 100 generations. In all NEA categories (I), (II), and (III), ADEANN (Ahmadizar et al., 2015) requires less computational effort than the other methods. In category (I), G3PCX (Deb et al., 2002) scores the highest classification accuracy, whereas ADEANN achieves the highest classification accuracy in categories (II) and (III). GEGA (Ahmadizar et al., 2015) generates the smallest neural networks in categories (I) and (III). ADEANN generates larger neural networks than the other methods in all three categories.

Tables 12, 13 and 14 statistically compare the performance of ADEANN and the other methods by the approach mentioned in Materials and Methods. ADEANN provides the best overall performance in categories (II) and (III) (Tables 13 and 14). In category I, GEGA provides the best overall performance (Table 12), but ADEANN outperforms G3PCX (Deb et al., 2002) and GA (Paz & Kamath, 2005). Fig. 7(a) and (b) plot the best and average fitnesses in the BC and KDDCUP'99 problems. Initially, ADEANN designs ANNs with low fitness. In subsequent generations, these fitnesses are improved by the fitness function, Eq. (5), which seeks the best topologies.

6.3. Comparison with other NEAs – Time series forecasting

The time-series forecasting of ADEANN and four other NEAs was evaluated on a benchmark of three time series, Passengers, Temperature and Dow–Jones (described in Table 6). The results are compared by the four statistical methods, described in Section 5.3: ARIMA, UCM, ADANN, and Forecast Pro. Table 15 summarizes the forecasting results for each time series obtained by the five methods. The methods are compared by the error metrics SMAPE (%) given by Eq. (12) and MSE given by Eq. (3). The best results are highlighted in boldface.

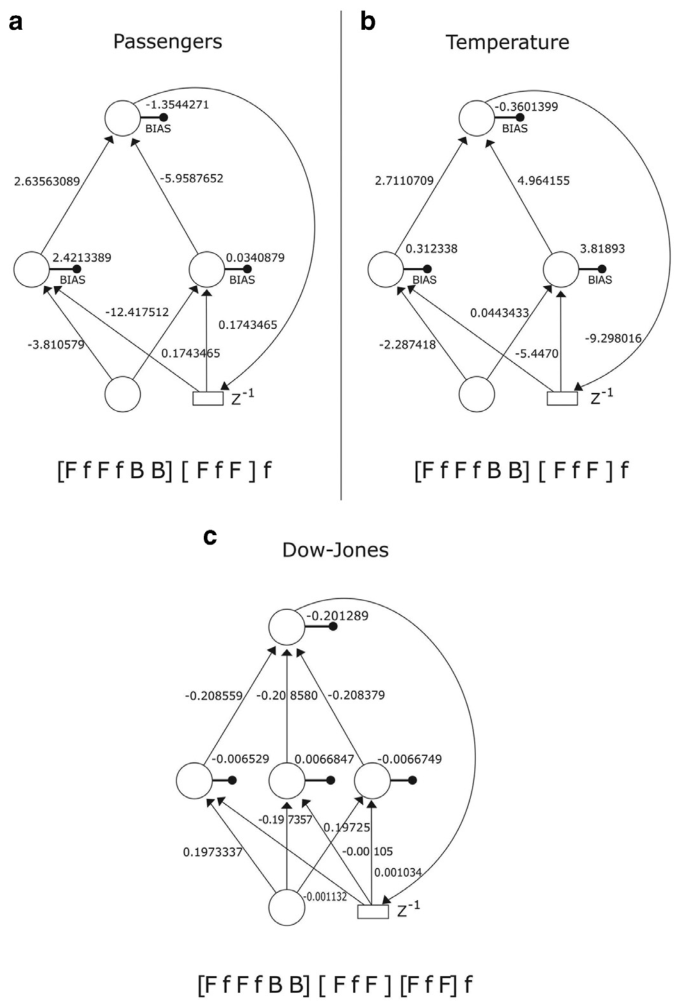


Fig. 8. ANNs returned by the search process (TSF).

From Table 15, we observe that ADEANN outperforms all other methods (ARIMA, UCM, the software tool Forecast Pro and ADANN (Donate et al., 2012)) in the Temperature and Dow–Jones forecasting. Therefore, ADEANN provides the best forecasting results in these two time series, The SMAPE (MSE) in the Temperature and Dow–Jones evaluation is 2.07% (0.17×10^{-2}) and 0.71% (0.26×10^{-4}), respectively. In the Passengers forecasting, ADEANN outperforms ARIMA, ADANN and the Forecast Pro software, and outperforms UCM by 0.03.

Table 16 statistically compares the time-series forecasting performances of ADEANN and the four existing methods. ADEANN outputs the best results for almost all of the datasets described in Table 6, and also provides the best overall performance. To clarify the closeness of the forecasting results to the real data, we plotted the forecasts of each time series on a graph. Zoomed-in plots of the Passengers, Temperature, and Dow–Jones forecast by ADEANN are shown in Fig. 10(a) , (b), and (c), respectively. The predicted values approximate the real values. Fig. 10(d) shows how the MSE depends on the number of neurons in the prediction of the Temperature time-series. The fitness function Eq. (5) selects the smallest possible ANN with the lowest MSE.

Selecting the fitness function is essential for defining the search domain of the GA. Panels (a) and (b) of Fig. 9 present the Pareto suboptimal and optimal fronts of Eqs. (4) and (5), respectively, and panels (c) and (d) present their respective contours. Closely spaced contour lines indicate more rapid change/steeper slopes. The closer proximity of the lines in Fig. 9(c) compared with Fig. 9(d) indicate a steeper surface in Fig. 9(a) compared with Fig. 9(b). Conversely, widely spaced contour lines Fig. 9(d) indicate relatively flatter regions. Determining appropriate coefficients for Eq. (4) is a problem-dependent, time-consuming process of the trial-and-error selection of A1 and A2. Eq. (5) realizes a much easier search process, thereby reducing the chances of trapping in local minima.

The ADEANN is further discussed in the next section.

7. Discussion

This section details other features of ADEANN.

(a) Minimal dependency on an expert— This paper proposes an automated system of pattern recognition and time-series forecasting, requiring only minimal knowledge from the user. The GP system (Rivero et al., 2010) has a number of parameters, three of which (the penalty coefficient, tree height, and maximum number of inputs to each neuron) must be set by an expert (Ahmadizar et al., 2015). However, ADEANN has only five parameters (listed in Table 4), which can be set by users.

(b) Compact DNA encoding — In our indirect genetic encoding (DNA encoding), a single gene is employed multiple times at different stages of the neuronal development. The reused genes (the phenotypic structures and rules) enable compact representations of complex phenotypes, and dimensional reduction of the solution space in the search process. To elucidate the compression provided by our IES, we consider the scalability reduction and number of production rules in comparison with other methods. In the methods of Kitano (1990) and Boozarjomehry and Svrcek (2001), the network sizes are quadratic functions of the chromosome length, given by N^2 and $(N(N+1)/2)$, respectively. For example, for the chromosome lengths listed in Table 7, which range from 180 to 516, the ANNs generated by the methods of (Kitano, 1990) and (Boozarjomehry & Svrcek, 2001) would contain (at most) 13–22 neurons and 18–31 neurons, respectively. In contrast, depending on the randomly generated NR, Eqs. (1) and (2) of ADEANN can generate much larger ANNs. Our method can manipulate chromosomes that are 30% shorter (180 bits versus 256) than those used in Graph Rewriting Grammar (GRG) (Paz & Kamath, 2005). In the IES of (Hornby & Pollack, 2002), the genotypes encode twenty rewrite rules of an L-system. Our DNA encoding system encodes a parametric L-system with only ten production rules, demonstrating that our IES is more compact than (Hornby & Pollack, 2002)'s version. DNA coding can arbitrarily represent developmental rules of any number and length (Lee et al., 2007). Additionally, the memory mechanism in our approach enables separate developmental pathways.

(Rivero et al., 2010) stored already existing neurons in a list, and referenced them by special operators. In contrast, our method generates graphs by a grammar encoded in the GA. Therefore, multiple neurons are implicitly referenced within the chromosomes of the GA, rather than formed into structures. Consequently, our architectures can grow indefinitely, regardless of the data structure.

(c) Implements implicit parallelism— Our GA is more biologically inspired than traditional GA. To better mimic biological evolution, our GA evolves a generative representation (a hypothetical DNA). Our NEA genotypes encode a set of twenty rewrite rules of an L-system. To model the constructive approach of real evolutionary processes, the L-rules in our method are automatically extracted by the GA. In this way, we control the number of firing times, i.e., the number of applications of each rule. The GA rule extraction in ADEANN is inspired by protein synthesis. This mechanism realizes a greater implicit parallelism than traditional GAs. In standard GAs, the original string is subjected to single-bit mutations, so each new string differs from its predecessor by only one bit. Consequently, each new string represents a new sample in only one schema region (Mitchell, 1998). Our approach implements

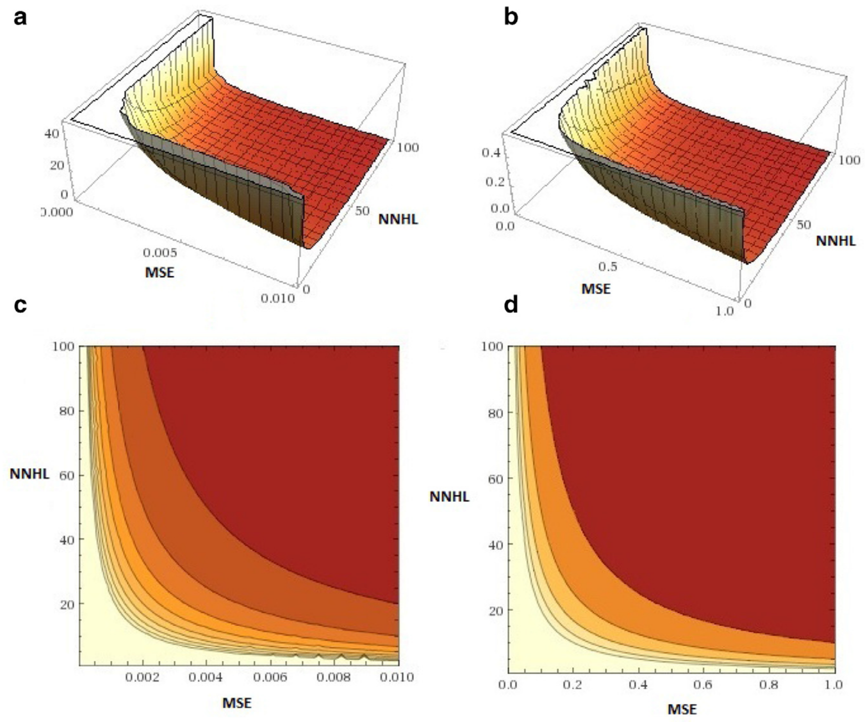


Fig. 9. Panels (a) and (b) of Fig. 9 present the Pareto suboptimal and optimal fronts of Eqs. (4) and (5), respectively, and panels (c) and (d) present their respective contours.

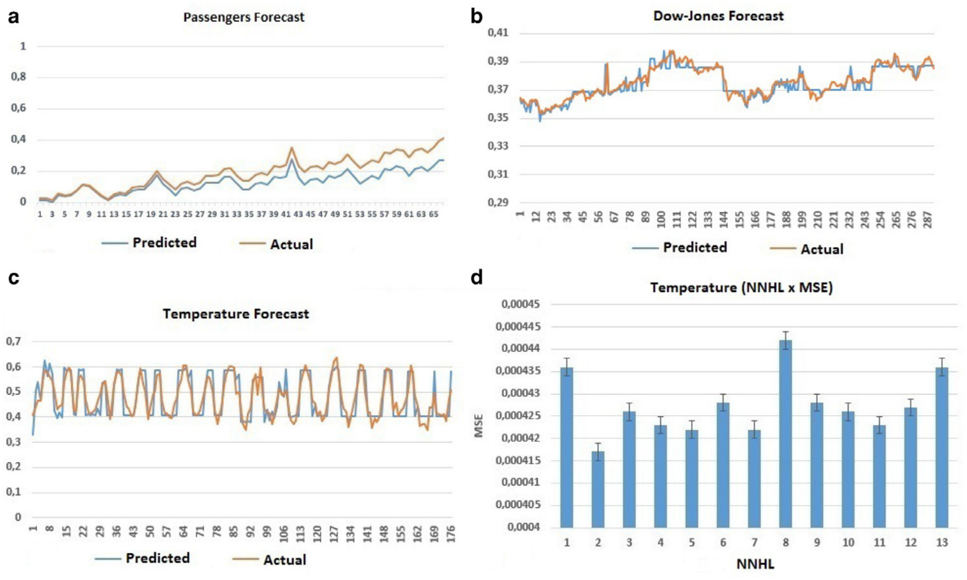


Fig. 10. Zoom-in of Passengers (a), Dow-Jones (b) and Temperature (c) forecast by ADEANN and (d) MSE versus number of hidden neurons in the Temperature forecast.

mutation by reading the chromosomes in several alternate positions (see Step 5 of function Rule-Extraction-with-GA). Thus, each new string is completely independent of the previous string, and new samples are autonomously assigned to each schema region. This mechanism accelerates the GA because it allows the simultaneous appearance of multiple desired schema in a given sample. Our GA is significantly faster than hill-climbing methods, which generate new samples by mutating single bits (or a small number of bits) during the search process.

(d) Network optimization — Our method also enables network optimization. As discussed in Section 5.1, the fitness function Eq. (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 7). 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.

(g) Performance— The performance and robustness of ADEANN were evaluated by *k*-fold cross validation. As shown in Section 6, ADEANN outperformed several existing methods in NEA categories

(II) (Table 13) and (III) (Table 14) and performed comparably to other methods in category (I) (Table 12). ADEANN required less computational effort than other methods in all categories (see Tables 9, 10 and 11). ADEANN achieved the highest average classification accuracy in 55.5% of cases. The time-series forecasting performance of ADEANN was also compared against those of ARIMA (Box & Jenkins, 1976), UCM, ADANN (Donate et al., 2012) and the forecasting software tool Forecast Pro. Table 15 summarizes the results of the three time-series benchmarks. ADEANN outperformed the other methods in all time-series (Passengers, Temperature and Dow–Jones), but yielded higher MSE than UCM in the Passengers time series. As clarified in Table 16, ADEANN obtained the best results in almost all datasets, and scored the highest overall performance. In addition, among the grammatical evolution methods–namely, ADEANN–GEGA (Ahmadizar et al., 2015), GE (Tsoulos et al., 2008) and GE-GP (Soltanian et al., 2013), ADEANN alone is bio-inspired, which contributes to its originality.

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 Eqs. (1) and (2) is randomly generated, the algorithm cannot generate all possible neuron topologies in the interval [*X*, *Y*] in a specific generation.

8. Conclusion

We developed an automatic pattern recognition and time-series forecasting system that requires minimal knowledge from a user. The user inputs the dataset or time series to be classified or forecasted and the number of elements to be generalized or forecasted into ADEANN. The algorithm outputs the classified or forecasted values.

To efficiently search the infinite space of architecture states, ADEANN represents the network topology in its chromosomes using a grammatical encoding system. This representation enables more directed searching. The system automatically implements a penalty approach that rewards economic ANNs. The resulting ANNs have strong generalization capacity and predictive ability and are easily implemented.

The performance of ADEANN was confirmed in several experiments on real-world classification and time-series forecasting. The results were statistically compared with those of well-known, state-of-the-art NEAs reported in the literature. Our NEA outperformed other methods and yielded the best overall performance.

NEAs that evolve recurrent neural networks are rarely found in the literature (Beer & Gallagher, 1992), (Hornby & Pollack, 2002). 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.

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.

Acknowledgment

The authors would like to thank Federal University of Pará.

References

Ahmadizar, F., Soltanian, K., AkhlaghianTab, F., & Tsoulos, I. (2015). Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm. *Engineering Applications of Artificial Intelligence, 39*(0), 1–13.

Alon, U. (2006). *An Introduction to Systems Biology: Design Principles of Biological Circuits* (1st ed.). (Chapman & Hall/CRC Mathematical and Computational Biology), Chapman and Hall/CRC.

Anderson, O. (1976). *Time series analysis and forecasting: the Box–Jenkins approach*. Butterworths, London.

Beer, R. D., & Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior, 1*, 91–122.

Blake, C. L., & Merz, C. J. (1998). *Uci repository of machine learning databases*.

Boozarjomehry, R. B., & Svrcek, W. (2001). Automatic design of neural networks structures. *Computers and Chemical Engineering*, (25), 1075–1088.

Box, G. E. P., & Jenkins, G. M. (1976). *Time series analysis: forecasting and control*. San Francisco: Holden-Day.

Cantu-Paz, E., & Kamath, C. (2005). An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. *Trans. Sys. Man Cyber. Part B, 35*, 915–927.

Castellani, M. (2013). Evolutionary generation of neural network classifiers-an empirical comparison. *Neurocomput, 99*, 214–229. doi:10.1016/j.neucom.2012.07.010.

Czajkowski, A., Patan, K., & Szymański, M. (2014). Application of the state space neural network to the fault tolerant control system of the plc-controlled laboratory stand. *Engineering Applications of Artificial Intelligence, 30*(0), 168–178.

Dasgupta, D., & McGregor, D. (1992). Designing application-specific neural networks using the structured genetic algorithm. In *Proceedings of the international conference on combinations of genetic algorithms and neural networks* (pp. 87–96).

Dawkins, R. (1986). *The blind watchmaker*. Longman Scientific and Technical.

de Campos, L. M., de Oliveira, R. C., & Roisenberg, M. (2015). Evolving artificial neural networks through l-system and evolutionary computation. In *Ijcnn* (pp. 1–9). IEEE. ISBN: 978-1-4799-1960-4.

de Campos, L. M. L., & de Oliveira, R. C. L. (2013). A comparative analysis of methodologies for automatic design of artificial neural networks from the beginnings until today. In *Paper presented at 1st BRICS countries congress (BRICS-CCI) and 11th brazilian congress (CBIC) on computational intelligence*. doi:10.1109/BRICS-CCI-CBIC.2013.81.

de Campos, L. M. L., Roisenberg, M., & de Oliveira, R. C. L. (2011). Automatic design of neural networks with l-systems and genetic algorithms - a biologically inspired methodology. In *Ijcnn* (pp. 1199–1206). IEEE. ISBN: 978-1-4244-9635-8.

Deb, K., Anand, A., & Joshi, D. (2002). A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation, 10*, 345–369.

Donate, J. P., Sanchez, G. G., & de Miguel, A. S. (2012). Time series forecasting. a comparative study between an evolving artificial neural networks system and statistical methods. *International Journal on Artificial Intelligence Tools, 21*.

Douglas, R. J., & Martin, K. A. C. (2007). Recurrent neuronal circuits in the neocortex. *Curr Biol., 3, 17*(13), R496–R500. PubMed PMID: 17610826.

Gauci, J., & Stanley, K. O. (2010). Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation, 22*, 1860–1898.

Gruau, F. (1994). Neural network synthesis using cellular encoding and the genetic algorithm. In *Unpublished doctoral dissertation, ecole normale superieure de lyon, l'universite claude bernard-lyon i*.

Hipel, K. W., & McLeod, A. I. (1994). *Time series modelling of water resources and environmental system*. (Elsevier, Amsterdam).

Hornby, G. S., & Pollack, J. B. (2002). Creating high-level components with a generative representation for body-brain evolution. *Artificial Life, 8*, 223–246.

Ilonen, J., Kamarainen, J.-K., & Lampinen, J. (2003). Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters, 17*, 93–105.

Kenneth, & Spears, W. M. (1992). A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence, 5*(1), 1–26.

Kitano, H. (1990). Designing neural networks by genetic algorithms using graph generation system. *Complex Systems Journal, 4*, 461–476.

Krömer, P., Platoš, J., & Snášel, V. (2014). Nature-inspired meta-heuristics on modern gpus: state of the art and brief survey of selected algorithms. *International Journal of Parallel Programming, 42*, 681–709.

Lee, D. W., Seo, S.-W., & Sim, K.-B. (2007). Evolvable neural networks based on developmental models for mobile rôbot navigation. *International Journal of Fuzzy Logic and Intelligent Systems, 7*, 176–181.

Li, X.-L., Jia, C., Liu, D.-X., & Ding, D.-W. (2014). Nonlinear adaptive control using multiple models and dynamic neural networks. *Neurocomputing, 136*, 190–200.

Lin, W.-Y., Lee, W.-Y., & Hong, T.-P. (2003). Adapting crossover and mutation rates in genetic algorithms. *J. Inf. Sci. Eng., 19*, 889–903.

Lindenmayer, A. (1968). Mathematical models for cellular interaction in development: parts i and II. *Journal of Theoretical Biology, 18*.

Miller, B. L., & Goldberg, D. E. (1996). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems, 9*, 193–212.

Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms* (pp. 379–384). Morgan Kauffman.

Mitchell, M. (1998). *An introduction to genetic algorithms*. Cambridge, MA, USA: MIT Press.

Niska, H., Hiltunen, T., Karppinen, A., Ruuskanen, J., & Kolehmainen, M. (2004). Evolving the neural network model for forecasting air pollution time series. *Engineering Applications of Artificial Intelligence, 17*, 159–167.

Oladele, R. O., & Sadiku, J. S. (2013). Genetic algorithm performance with different selection methods in solving multi-objective network design problem. *International Journal of Computer Applications, 70*, 5–9.

Risi, S., & Stanley, K. O. (2012a). An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life, 18*, 331–363.

Rivero, D., Dorado, J., Rabual, J., & Pazos, A. (2010). Generation and simplification of artificial neural networks by means of genetic programming. *Neurocomputing, 73*, 3200–3223.

Rumelhart, D., & McClelland, J. L. (Eds.) (1986). *Parallel distributed processing: explorations in the microstructure of cognition*: 1. Foundations. Cambridge, MA: MIT Press.

Sanchez, D., & Melin, P. (2014). Optimization of modular granular neural networks using hierarchical genetic algorithms for human recognition using the ear biometric measure. *Engineering Applications of Artificial Intelligence, 27*, 41–56.

Soltanian, K., Tab, F., Zar, F., & Tsoulos, I. (2013). Artificial neural networks generation using grammatical evolution. In *Icee* (pp. 1–5). IEEE. INSPEC Accession Number: 13767754.

Stanley, K. O., D'Ambrosio, D. B., & Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life, 15*, 185–212.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation, 10*(2), 99–127.

Stanley, K. O., & Miikkulainen, R. (2003). A taxonomy for artificial embryogeny. *Artificial Life, 9*, 93–130. doi:10.1162/106454603322221487.

Tsoulos, I., Gavrilis, D., & Glavas, E. (2008). Neural network construction and training using grammatical evolution. *Neurocomputing, 72*, 269–277. doi:10.1016/j.neucom.2008.01.017.