Neurocomputing 121 (2013) 274–289

Contents lists available at ScienceDirect

Neurocomputing

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

Fast learning neural networks using Cartesian genetic programming

Maryam Mahsal Khan [a], Arbab Masood Ahmad [b], Gul Muhammad Khan [a,*], Julian F. Miller [c]

[a] Department of Electrical Engineering, UET Peshawar, Pakistan
[b] Department of Computer System Engineering, UET Peshawar, Pakistan
[c] Department of Electronics, University of York, UK, YO10 5DD

ARTICLE INFO

Article history:
Received 17 August 2011
Received in revised form
24 December 2012
Accepted 27 April 2013
Communicated by H. Zhang
Available online 15 May 2013

Keywords:
Artificial neural network
Pole balancing
Breast cancer
Neuroevolution
Recurrent networks

ABSTRACT

A fast learning neuroevolutionary algorithm for both feedforward and recurrent networks is proposed. The method is inspired by the well known and highly effective Cartesian genetic programming (CGP) technique. The proposed method is called the CGP-based Artificial Neural Network (CGPANN). The basic idea is to replace each computational node in CGP with an artificial neuron, thus producing an artificial neural network. The capabilities of CGPANN are tested in two diverse problem domains. Firstly, it has been tested on a standard benchmark control problem: single and double pole for both Markovian and non-Markovian cases. Results demonstrate that the method can generate effective neural architectures in substantially fewer evaluations in comparison to previously published neuroevolutionary techniques. In addition, the evolved networks show improved generalization and robustness in comparison with other techniques. Secondly, we have explored the capabilities of CGPANNs for the diagnosis of Breast Cancer from the FNA (Finite Needle Aspiration) data samples. The results demonstrate that the proposed algorithm gives 99.5% accurate results, thus making it an excellent choice for pattern recognitions in medical diagnosis, owing to its properties of fast learning and accuracy.

The power of a CGP based ANN is its representation which leads to an efficient evolutionary search of suitable topologies. This opens new avenues for applying the proposed technique to other linear/non-linear and Markovian/non-Markovian control and pattern recognition problems.

1. Introduction

Artificial neural networks (ANNs) not only have the ability to extract information from complex data but also have the potential to resolve linear/non-linear problems in fields such as chemical processes, control, robotics, pattern recognition, computer vision, oil and gas, etc. [1–3].

Control Systems are conventionally developed by constructing a mathematical model that represents all dynamics of the system [4]. However, some systems that are complex and non-linear cannot be mathematically modeled, as there are no standard and conventional procedures to represent them. Thus the efficient design and development of a controller becomes difficult. Intelligent control is an unconventional process that provides the flexibility of generating abstract models without any indication of hidden dynamics of the system.

One training method used in ANNs is a genetic algorithm. This is broadly known as neuroevolution. For decades, the performance of the neuroevolutionary algorithms has been tested on the non-linear control problem e.g. developing linear and non-linear controller of a standard benchmark problem 'the inverted pendulum' for multiple scenarios of poles, optimum localization of mobile robots, auto-pilot helicopter or aircraft controller, automobile crash warning system, rocket control, routing over a data network, coordinating multi-rover systems, time-series prediction, lung sound detection, speech recognition, chemical processes and manufacturing, and the octopus arm task [5–11]. Many of the developed controllers have been found efficient and useful when compared to the controllers developed by conventional methods.

Genetic Programming is a form of automatic program induction where evolutionary algorithms are used to build computer programs and complex data structures [12–14]. In this paper, we are using a graph based form of genetic programming called Cartesian genetic programming (CGP) [15–17]. CGP has been explored in a range of diverse application domain and has shown to produce competitive performance [16,18–22]. A powerful aspect of CGP is its representation of graphs coupled with a high degree of genetic redundancy.

We have adapted CGP for representation and evolution of neural networks. We call our neuroevolutionary technique, CGP artificial neural networks (CGPANNs). Both feedforward and recurrent representations are examined. It is applied on a standard benchmark problem—pole balancing ranging from a simple setup to extremely difficult versions. The results obtained demonstrate that

* Corresponding author. Tel.: +92 91 9216796; fax: +92 91 9216663.
E-mail addresses: gk502@nwfpuet.edu.pk, engineergul@yahoo.com (G. Muhammad Khan).

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2013.04.005

the proposed technique has a fast learning capability as it consistently outperforms all the previous methods explored to date.

The CGPANN technique is also applied to the problem of detecting breast cancer. Features from breast mass are extracted using fine needle aspiration (FNA) and the information is applied as input to CGPANN. FNA data available at the Wisconsin Diagnostic Breast Cancer web site is used for training and testing the network capabilities. The system developed produces fast and accurate results when compared to contemporary work done in the field. The error of the model comes out to be as low as 1% for Type-I (wrongly classifying benign samples as malignant) and 0.5% for Type- II (wrongly classifying malignant samples as benign). The paper is organized as follows. Section 2 is an overview of neuroevolution and the different algorithms developed so far. Section 3 describes the background information on Cartesian genetic programming. Section 4 describes the neuroevolutionary algorithm based on Cartesian genetic programming in detail. Section 5 describes the algorithm applied on the standard benchmark problems i.e. single and double pole balancing tasks along with simulation, analysis and results. Section 6 represents in detail the performance of the algorithm on the diagnosis of breast cancer. Section 8 concludes with discussions and future work.

2. Neuroevolution

Artificial neural networks (ANNs) are computational systems that map complex relationships between inputs and outputs. They are made up of interconnected nodes (neurons) and weighted connections. The properties of these nodes are inspired by biological neurons, and can exhibit complex global behaviour.

A number of evolutionary algorithms have been applied in the past decade to evolve either weights, topology or both of the parameters of artificial neural networks also known as TWEANN (Topology and Weight Evolving Artificial Neural Network). TWEANN algorithms can be 'direct' encoding schemes with the genotype representing all connections and nodes that appear in the phenotype, or they can be 'indirect' where the genotype specifies the rules for constructing the phenotype. Direct encodings are generally better at fine tuning and generating compact architectures [23–30], whereas indirect encodings are better at finding a particular type of ANN architecture (i.e. topology) [23,31–34].

Yao reviewed different combinations of ANNs and evolutionary algorithms (EAs) that evolved ANN connection weights, architectures, learning rules, and input features [35]. He used both direct and indirect encoding scheme and pointed out that the direct encoding scheme of ANN architectures is very good at fine tuning and generating a compact architecture, whereas the indirect encoding scheme is suitable for finding a particular type of ANN architecture quickly. In further analysis, he identified that separating the evolution of architectures and connection weights can cause fitness evaluation to mislead evolution, whereas simultaneous evolution of ANN architectures and connection weights produces better results [35].

In symbiotic adaptive neural evolution (SANE), the neuron population along with the representations of network topologies are evolved together. SANE has successfully been applied to pole balancing task obtaining the desired behaviour within relatively few evaluations [10]. ESP referring to enforced subpopulations is an extension to SANE where instead of one population of neurons, a subpopulation of hidden layer neurons is evolved. This has produced better results than SANE [36].

In conventional neuroevolution (CNE) a genotype represents the whole neural network similar to Wieland's algorithm [37]. It uses rank selection and burst mutation. The conventional algorithm has advantages over ESP as it evolves genotypes at the entire network level rather than neuron level, thus allowing potential global solutions to be found for a predefined network topology and size.

Cooperative synapse neuroevolution (CoSyNE) evolves neural network at the level of synaptic weights only. The CoSyNE approach has been shown to out-perform all the previous approaches on pole balancing problem [9,36].

Stanley presented a TWEANN, known as neuroevolution of augmenting topologies (NEAT) [38]. NEAT was shown to perform faster than many other neuro-evolutionary techniques. Unlike a conventional neural network whose topology is defined by the user, NEAT allows the user to evolve the network topology. Thus the complexity of the network can change. The algorithm is notable in that it evolves both network weights and structure, and efficiently balances between the fitness and diversity of evolved solutions. NEAT has also produced competitive results on pole balancing problem.

Cooperative co-evolution networks (COVNET) focus on the evolution of co-adapted subcomponents. Instead of evolving complete networks only subnetworks are evolved. COVNET uses this technique and has been shown to produce better generalization and smaller networks [11].

Tsoy and Spitsyn introduced a neuroevolutionary algorithm known as NeVA and applied it to the pole balancing problem. In NeVA, adaptive mutation of connections and weights of a network is performed [39].

Kazuhiro also looked at the pole balancing problem using MBEANN (Mutation Based Evolutionary ANN), which is a TWEANN algorithm where evolution is based on mutation only [40].

Continuous time recurrent neural networks (CTRNNs) are derived from dynamical neuron models known as leaky integrators and their behaviour is modeled mathematically using a system of differential equations. It is an intermediate step between sigmoidal and spiking neuron. Angeline et al. [41] evolve both the weights and topology of CTRNNs and argued that this provides a number of advantages over traditional approaches. In [42] CTRNNs are evolved using NEAT and applied on the pole balancing problem. This produced better results than the standard NEAT algorithm.

The HyperNEAT algorithm consists of a neural network called the substrate comprising nodes that are connected together [43,44]. The weights of these connections are encoded using another network called CPPN (Compositional Pattern Production Network) which is evolved using NEAT (Neuro-Evolution of Augmenting Topology) algorithm. The inputs to CPPN are the coordinates of the nodes of the ANN while the output is the weight of the connection between these nodes. The output is in the form of a pattern in a four dimensional Hyper cube, where pairs of dimensions correspond to the coordinates of the source and target neurons. The HyperNEAT algorithm has features in common with biological networks of neurons in which neurons corresponding to neighboring sensors are located closer together. Buk et al. [45] compared the evolutionary computation technique of Hyper NEAT with the modified form of it which they named HyperGP. In HyperGP the CPPN is evolved using Genetic Programming. While the desired fitness reached by evolving the weights of the ANN using NEAT algorithm in CPPN takes 92 generations, the same fitness is achieved with GP for the same population size in only 20 generations. Risi et al. [46] developed a new form of HyperNEAT called ES-HyperNEAT in which the number and location of nodes is decided by the algorithm. This differs from the original HyperNEAT which consists of fixed substrate where the number of hidden nodes is decided a priori, while only the connection weights are evolved by CPPN. In ES-HyperNEAT the locations of these nodes are determined by finding the variance of weights throughout the substrate. Nodes are automatically removed from the substrate at locations that have a variance lower than a threshold. Thus the algorithm takes care of the number of nodes and their associated weights.

Pujol and Poli used a form of Genetic Programming (GP) called Parallel Distributed GP (PDGP) to evolve weights, topology and

activation functions of ANNs. PDGP is a graph-based representation of programs [47]. They tested the system on a variety of computational problems, including the development of a neural controller for the pole balancing problem with promising results [48].

Interestingly, recently Poli and Graf have shown that there are "Free Lunches for Neural Network Search" so that "…not all algorithms are equally good at finding neural networks that solve problems under all possible performance measures: a superior search algorithm for this domain does exist." [49]. This strongly motivates the search for more effective algorithms for designing ANNs.

3. Cartesian genetic programming

CGP programs are represented in the form of directed acyclic graphs. These graphs are represented as a two dimensional grid of computational nodes. The genes that make up the genotype in CGP are integers that represent where a node gets its data, what operations the node performs on the data, and where the output data required by the user is to be obtained. When the genotype is decoded, some nodes may be ignored. This happens when node outputs are not used in the calculation of output data. We refer to such nodes and their genes as 'non-coding'. We call the graph that results from the decoding of the genotype, a phenotype. The genotype in CGP has a fixed length. However, the size of the phenotype (in terms of number of computational nodes) can be anything from zero nodes to the number of nodes defined in the genotype. The types of computational node functions used in CGP is decided by the user and are listed in a function look-up table. Later, we discuss how CGP can be used to evolve neural networks, and use two possible functions sigmoid or hyperbolic tangent. In CGP, each node represents a particular function and is encoded by a number of genes. The *function gene* is the address of the computational node function in the function look-up table. The remaining node genes say where the node gets its data from. We call these *connection genes*. Nodes take their inputs in a feed forward manner from either the output of nodes in a previous column or from a program input. The number of connection genes a node has is chosen to be the maximum number of inputs (often called arity) that any function in the function look-up table has. The program data inputs are given the absolute data addresses 0 to n_i-1 where n_i is the number of program inputs. The data outputs of nodes in the genotype are given sequentially addresses, column by column, starting from n_i to n_i+L_n-1 where L_n is the user-determined upper bound of the number of nodes. The general form of a Cartesian Genetic Program is shown in Fig. 1. In general, there may be a number of output genes (O_i) which specify where the program outputs are taken from. Each of these is an address of a node where the program output data is taken from. Nodes in columns cannot be connected to each other. The graph is directed and feed-forward this means that a node may only have its inputs connected to either input data or the output of a node in a previous column. The structure of the genotype is shown below the schematic in Fig. 1.

CGP has three parameters that are chosen by the user. These are *number of columns*, *number of rows*, *levels-back*. The latter controls the connectivity of the graph encoded. If it is one, a node can only get its inputs from a node in the column on its immediate left or from a primary input. If one wishes to allow nodes to connect to any nodes on their left then *levels-back* is set to the number of columns.

The values that genes can take (i.e. alleles) in CGP are highly constrained. When genotypes are initialized or mutated, these constraints should be obeyed. First of all, the alleles of function genes must take valid address values in the lookup table of primitive functions. In feed forward graphs the values taken by the connection genes must be such that they refer to the outputs of nodes in previous columns (or inputs). They also must obey the constraints imposed by the *level-back* parameter. Note that connection to inputs are usually allowed to disobey this constraint.

3.1. Example of CGP genotype and phenotype

We give a simple example of CGP in Fig. 2 where Fig. 2(a) shows a 2×2 (rows $\times$ columns) genotype for a problem with 2 binary inputs and 1 binary output. The functions used are logical 'OR' and logical 'AND'. The user defined parameter (arity) indicating the number of node inputs is set to 2 in this example. Fig. 2(b) shows the graphical representation of the genotype in Fig. 2 and shows how the genotype in (a) is decoded. The output gene 5 dictates that the node whose output label is 5 is selected as the output of the network. The genotype represents the equation, $Y=x_0\cdot x_1+x_1$, where '$\cdot$' represents the logical AND operation and '+' the logical OR. In the example nodes with outputs 2 and 4 are both non-coding.

3.2. Evolution of CGP genotypes

The mutation operator used in CGP is a point-mutation operator. In point mutation the allele at a randomly chosen gene location is changed to another valid random value. If a function gene is chosen for mutation, then a valid value is the address of any function in the function set. Whereas, if an input gene is chosen for mutation, then a valid value would be the address of the output of any previous node in the genotype or of any program input. Also, a valid value for a program output gene is the address of the output of any node in the genotype or the address of a program input. The number of genes in the genotype that can be mutated in a single application of the mutation operator is defined by the user, and is normally a percentage of the total number of genes in the genotype. We refer to the latter as *mutation rate* and use the symbol μ_r to represent it.

A variant on a simple evolutionary algorithm known as a $1+\lambda$ evolutionary algorithm [50] is widely used for CGP. Usually λ is chosen to be 4. This has the form shown in procedure 1. On line 10 of Procedure 1 there is an extra condition that when offspring genotypes in the population have the same fitness as the parent and there is no offspring that is better than the parent. In that case an *offspring* is chosen as the new parent. This is a very important feature of the algorithm which makes good use of redundancy in CGP genotypes. Inactive genes have a neutral effect on genotype fitness. CGP genotypes are dominated by redundant genes. The influence of redundancy in CGP has been investigated in detail [17,51–53] and has been shown to be extremely beneficial to the efficiency of evolutionary process on a range of test problems.

Algorithm 1. The (1+4) evolutionary strategy.

1: **for all** i such that $0\le i<5$ **do**
2: Randomly generate individual i
3: **end for**
4: Select the fittest individual, which is promoted as the parent
5: **while** a solution is not found **or** the generation limit is not reached **do**
6: **for all** i such that $0\le i<4$ **do**
7: Mutate the parent to generate offspring i
8: **end for**
9: Generate the fittest individual using the following rules:
10: **if** an offspring genotype has a better or *equal* fitness than the parent **then**
11: Offspring genotype is chosen as fittest
12: **else**
13: The parent chromosome remains the fittest
14: **end if**
15: **end while**

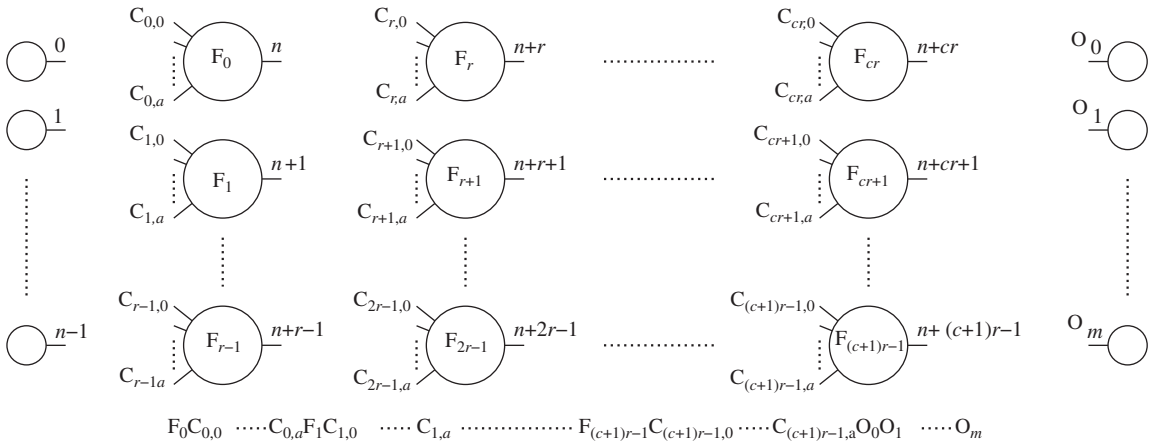

Fig. 1. General form of CGP. It is a grid of nodes whose functions are chosen from a set of primitive functions. The grid has n_c columns and n_r rows. The number of program inputs is n_i and the number of program outputs is n_o. Each node is assumed to take as many inputs as the maximum function arity a. Every data input and node output are labeled consecutively (starting at 0) which gives it a unique data address which specifies where the input data or node output value can be accessed (shown in the figure on outputs of inputs and nodes).

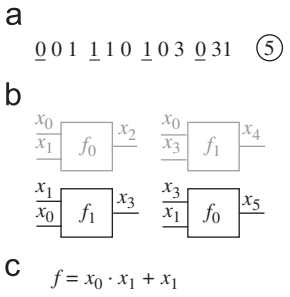

Fig. 2. (a) CGP based Genotype for a problem with 2 binary inputs x_0 and x_1. The functions genes are underlined. The single program output is taken from the last gene (circled). (b) Decoding step: Some nodes are not connected in the output path. The final phenotype is a logical equation giving the output $f = x_6$ as a function of the two program inputs. In the equation, + represent OR and · represents AND. (a) Genotype, (b) Decoding and (c) Phenotype.

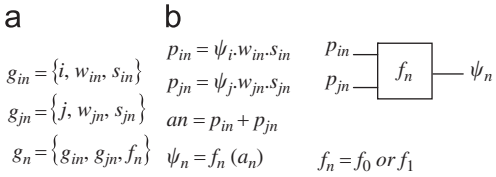

Fig. 3. (a) FCGPANN node with two inputs. The genes g_{in} and g_{jn} associated with node, n, are triples of values: Connection genes i and j indicate that ψ_i and ψ_j are the input signals to a neuron. ψ_i denotes an external input, x_i when $i < n_i$ and a node output otherwise. w_{in} and w_{jn} are the corresponding weights and s_{in} and s_{jn} are switch values (taking the values 0 or 1) (b) The argument a_n of the node function is the arithmetic sum of products. The number of products depends on the node arity (in this case it is two). Each node product is the product of the node input and the corresponding weights and switch values. The FPCGPANN node functions, f_n can be one of two possible functions f_0 is the tanh and f_1 is the *sigmoid*.

4. Cartesian genetic programming-based artificial neural networks

This section describes a neuroevolutionary algorithm that exploits the representation of CGP in generating artificial neural architecture.

Feed forward and recurrent architectures are evolved based on Cartesian genetic programming hereby known as FCGPANN and RCGPANN. Both the architectures follow a direct encoding strategy in which topology, weight and functions are encoded in one genotype. This is evolved to obtain a good topology, possible weights and combination of functions. So the genotype described in Section 3 has been augmented with genes representing the weights of connection and switch genes representing whether or not the connection is considered present. Also the node functions have been chosen to be one of two, a sigmoid or a hyperbolic tangent. When weights are mutated, a new real number random value is generated within the range [−1,1]. When a switch gene is mutated its binary value is inverted.

In general TWEANN algorithms are either constructive or destructive [35]. FCGPANN and RCGPANN are both constructive and destructive algorithms. The network derived constitutes neurons that are not fully connected. They begin with random topological features and evolution removes and adds new features. It does so by mutating functions, connections, weights, switches and outputs. Mutations to connections and switches can make a large range of changes to the topology of the encoded ANN. The mutations may include previously unconnected neurons (together with their weighted connections) or it may disconnect groups of neurons. The various topologies that are explored by evolution using the CGP representation can be highly variable. In addition the program input(s) are not necessarily supplied to every neuron in the input layer (some inputs can be supplied multiple times to the same neuron), this is different from traditional ANN architectures.

4.1. Feed forward Cartesian genetic programming evolved ANN

The FCGPANN genotype consists of the genes of nodes, where each node corresponds to a neuron of ANN and an output gene that determines which neuron provides the network output. The node includes input signals, weights, switches and function as shown in Fig. 3(a). As in CGP, inputs to nodes can be primary network inputs or inputs from the previous nodes (ones in previous columns). When the genotype is decoded we draw an array of nodes (as in Fig. 3(b)) in which we indicate where the signal inputs to the node come from. The symbols for the weights and switches can be inferred from the indices of the node inputs. So, for instance, the weight of the connection between the node with output ψ_n and the node providing its input ψ_m is w_{mn}. Note that if m is less than the number of external inputs then ψ_m denotes an external input, which we denote by x_m.

Input signals, weight and switch genes corresponding to an input are multiplied for all of the connected inputs and a summation is taken for all products. The number of products is dictated by the arity of the node function, which is a user defined parameter. The summation becomes the argument of the node

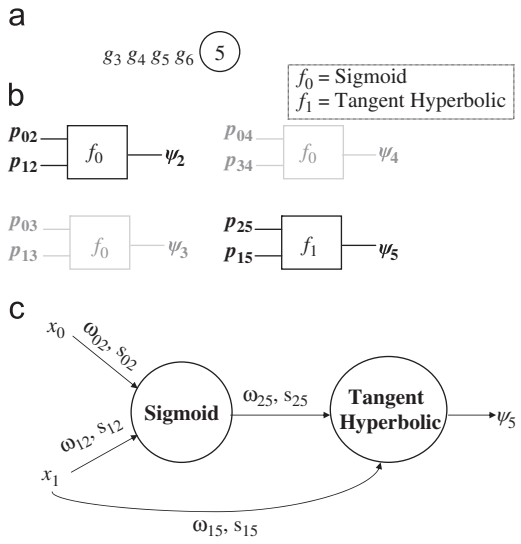

Fig. 4. (a) FCGPANN based genotype for a problem with two inputs x_0 and x_1. There are two primitive functions, f_0 and f_1 which represent sigmoid and tangent hyperbolic respectively. The application has a single program output. In the genotype, this is taken from the last gene (encircled). (b) Decoding step for genotype in (a): the output of each node is labeled consecutively from the program inputs. Some nodes are not connected in the output path, these are shown in grey. (c). The phenotype in the form of neural network for the genotype in (a).

neuron function. In the work we report here we used only two functions, either hyperbolic tangent or sigmoid, but in practice other functions can be easily accommodated. Using different functions helps not only in quickly finding solution but also assists in generating diverse solutions to the same problem.

Fig. 4(a) is a genotype of an FCGPANN structure with two inputs x_0 and x_1. Each g_i represents a triple of genes for each input to the neuron. The output of the network is taken from the node labelled 5. Fig. 4(b) shows the decoding of the genotype in Fig. 4(a). ψ_5 corresponds to the node labelled 5 which is the output of the genotype. This node only refers to one previous node (node 2—represented as ψ_2). Thus only two nodes are used (active) in the computation of the output. Fig. 4(c) displays the phenotype corresponding to the genotype. Eqs. (1) and (2) are the mathematical representation of the network in Fig. 4(c) where Eq. (2) shows the output of the FCGPANN network:

$$\psi_2 = sigmoid(x_0 \cdot w_{02} \cdot s_{02} + x_1 \cdot w_{12} \cdot s_{12}) \quad (1)$$

$$\psi_5 = \tanh(\psi_2 \cdot w_{25} \cdot s_{25} + x_1 \cdot w_{15} \cdot s_{15}) \quad (2)$$

4.2. Recurrent Cartesian genetic programming evolved ANNs

In order to capture a larger domain of systems that are dynamic and non-linear, the need for recurrent networks becomes essential. This section describes how recurrent neural networks can be evolved using the representation of Cartesian genetic programming.

The recurrent network produced is based on one of the earliest networks namely the 'Jordan Network' [54]. In this network the outputs are fed back into the input layer through sets of extra inputs called the 'state units'. The number of state units is equivalent to the number of outputs in the system. The connection from the output to the state unit has a fixed weight of +1. Learning takes place by updating the weights and switches between input, hidden and the output layer. It is to be noted that the state unit is only connected to the first layer.

In this paper a variant of Jordan network is proposed which involves connecting state unit inputs to any layer. Moreover a non-linear activation function is used at the state unit. The modified state unit is then tested by mutating different combination of weights and switches as explained in detail in Section 5.3. Like the FCGPANN genotype, the RCGPANN genotype also consists of nodes that represent neurons of ANN. The only difference is that the output of the network is fed back as an additional input. In this case, we assigned the node arity to be the number of primary inputs plus one. RCGPANNs differ from Jordan networks as we relaxed the condition that the state unit input is only used by the first layer (column) of nodes. Nodes in other layers (i.e. other than 1) can have zero or more feedback inputs. Moreover we have used a hyperbolic tangent function as our activation function in the state unit which is explained further in the example in Fig. 5.

Fig. 5(a) represents an RCGPANN genotype with primary inputs x_0, x_1 and a feedback input x_2. The node labelled 6 is the output of the network. This node serves as a feedback to the system. The initial value of this node is taken as zero. Fig. 5(b) displays the decoded genotype of Fig. 5(a). ψ_6 represents the output node. The node consists of input from a previous node ψ_3, a primary input of the system x_1 and feeback input x_2. From Fig. 5 it is clear that the state unit has two inputs: one is the output of the system and the other is the previous output of the state unit. This is mathematically expressed in Eq. (3) for the given genotype in Fig. 5(a) as

$$x_2 = \tanh(x_6 + x_2(t-1)) \quad (3)$$

The graphical representation of the decoded genotype is expressed in Fig. 5(c). This phenotype can be mathematically described using Eqs. (4) and (5).

$$\psi_3 = sigmoid(x_0 \cdot w_{03} \cdot s_{03} + x_1 \cdot w_{13} \cdot s_{13} + x_2 \cdot w_{23} \cdot s_{23}) \quad (4)$$

$$\psi_6 = \tanh(\psi_3 \cdot w_{36} \cdot s_{36} + x_1 \cdot w_{16} \cdot s_{16} + x_2 \cdot w_{26} \cdot s_{26}) \quad (5)$$

The proposed neuroevolutionary algorithms (FCGPANN and RCGPANN) are investigated to check their capabilities as detailed in Section 5.

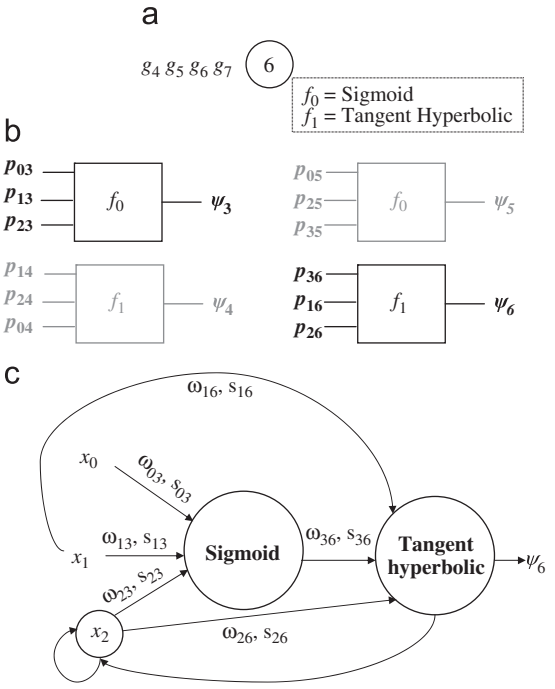

Fig. 5. (a) RCGPANN based genotype for a problem with two inputs x_0 and x_1. Input x_2 represents the recurrent input i.e. the output of the system is fed back as an input. The two primitive functions, f_0 and f_1 represent sigmoid and tangent hyperbolic respectively. The single program output is taken via the last gene (encircled). (b) Decoding step of genotype in (a). The final phenotype in (c), a recurrent neural network structure of the genotype in (a).

5. Case study-I: Pole balancing

Pole balancing is a standard benchmark problem from the field of control theory and artificial neural networks. The aim is to design controllers for unstable and non-linear systems [55,56]. Single pole balancing consists of a single pole while double pole balancing requires balancing two poles. The poles are attached to the wheeled cart by a hinge. The length of the track the cart can move on is limited to $-2.4 < x < +2.4$. The objective is to apply force 'F' to the cart in such a way that the angle of the pole should obey the following constraints: $-12^\circ < \theta_1 < +12^\circ$ for the single pole and $-36^\circ < \theta_{1,2} < +36^\circ$ for the double poles. Also the cart is confined to a finite length track (see Tables 1 and 2). The controller has to balance the pole(s) for approximately 30 min which corresponds to 100 000 time steps. Thus the neural network must apply force to the cart to keep the pole(s) balanced for as long as possible where any force 'F' greater than or equal to zero is assumed to correspond to +10 N and 'F' less than zero corresponds to −10 N. The system inputs are the pole-angle(s) θ_i, the angular velocity of pole(s) $\dot{\theta}_i$, the position of cart x and the velocity of cart $\dot{x}$. Where for single pole balancing i is 1 and for double pole balancing i is 1 or 2. Fig. 6 shows the single and double pole balancing scenarios.

Tables 1 and 2 display the standard numerical values used for simulating the single and double pole-balancing problems.

Table 1
Parameters for single pole balancing task.

Parameters	Value
Mass of cart (m_c)	1 Kg
Mass of Pole (m_{p1})	10 Kg
Length of Pole (l_1)	0.5 m
Width of the Track (h)	4.8 m

Table 2
Parameters for double pole balancing task.

Parameters	Value
Mass of cart (m_c)	1 Kg
Mass of Poles (m_{p1},m_{p2})	0.01,10 Kg
Length of Poles (l_1,l_2)	0.05,0.5 m
Friction of the Poles (μ_p)	0.000002
Friction of the cart (μ_c)	0.0005
Width of the Track (h)	4.8 m

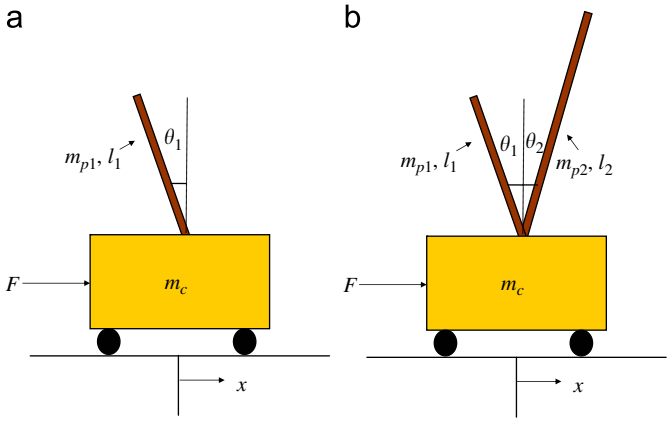

Fig. 6. (a) Single pole balancing, (b) double pole balancing.

Eqs. (6), (7), (8) and (9) are used to compute the effective mass of poles, acceleration of poles, acceleration of cart, and effective force on each pole, respectively. Eqs. (10), (11), (12) and (13) calculate the next state of angle of poles, velocity of poles, position of cart and velocity of the cart , respectively [55]:

$$\hat{m}_i = m_i\left(1-\frac{3}{4}\cos^2\theta_i\right) \tag{6}$$

$$\ddot{\theta}_i = -\frac{3}{4l_i}\left(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}\right) \tag{7}$$

$$\ddot{x} = \frac{F-\mu\,\mathrm{sgn}(\dot{x}) + \sum_{i=0}^{N}\hat{F}_i}{M+\sum_{i=0}^{N}\hat{m}_i} \tag{8}$$

$$\hat{F}_i = m_i l_i \theta_i^2 \sin\theta_i + \frac{3}{4}m_i\cos\theta_i\left(\frac{\mu_{pi}\theta}{m_i l_i} + g\sin\theta_i\right) \tag{9}$$

$$x[t+1] = x[t] + \tau\dot{x}[t] \tag{10}$$

$$\dot{x}[t+1] = \dot{x}[t] + \tau\ddot{x}[t] \tag{11}$$

$$\theta_i[t+1] = \theta_i[t] + \tau\dot{\theta}_i[t] \tag{12}$$

$$\dot{\theta}_i[t+1] = \dot{\theta}_i[t] + \tau\ddot{\theta}_i[t] \tag{13}$$

The proposed neuroevolutionary algorithm is used in the following cases:

5.1. Single pole: Markovian and Non-Markovian

For single pole Markovian case, the network or controller has access to all four inputs (x, $\dot{x}$, θ_1, $\dot{\theta}_1$) while for the non-Markovian case the number of inputs is reduced to two (x, θ_1). Single pole balancing is a linear problem.

5.2. Double poles: Markovian and Non-Markovian

In double pole balancing, for the Markovian case the network has access to all six inputs (x, $\dot{x}$, $\theta_{1,2}$, $\dot{\theta}_{1,2}$) while for the non-Markovian case the number of inputs is reduced to three (x, θ_1, θ_2). Double pole balancing is a non-linear problem.

In both the cases, the networks are generated with initial input values of zero or with random values of $-0.25\text{ rad} < \theta_1 < 0.25\text{ rad}$ and $-2.4 < x < +2.4$ for single pole and $-0.6\text{ rad} < \theta_{1,2} < 0.6\text{ rad}$ and $-2.4 < x < +2.4$ for double poles.

5.3. Experimental setup

For the two cases, various FCGPANN and RCGPANN genotypes of different network sizes with randomly generated connections, weights, outputs and switches are created. Weights are randomly produced between −1 and +1. The activation functions used are sigmoid and tangent-hyperbolic. The percentage mutation rate of $\mu_r = 10$ and $\mu_r = 20$ are used for all models. In all the experiments *levels-back* was set equal to the number of columns. The performance of the algorithm is based on the average number of balancing attempts (evaluations) in 50 independent evolutionary runs that resulted in the pole being balanced for over 100 000 time steps. It is to be noted that all runs ended in the pole(s) being balanced.

For the RCGPANN networks evolution of the genotypes is performed in four different ways to check the individual and combined performance of the state unit switches and weight:

- In the first method (*SW′*), we do not allow the weights of connections of the state unit to be mutated.

- In the second method (*S′W*), we do not allow switching genes referring to the state unit to be changed, but we allow weights to change by mutation.
- The third case (*SW*) we allow both weights and switching genes to be altered by mutation.
- The final case (*S′W′*) allows neither the weights nor switching genes referring to the state unit to be changed by mutation.

5.3.1. Single pole: Markovian and non-Markovian cases

We chose the number of evolved inputs that a neuron node can have to be closely related to the number of network inputs. In the Markovian case, the networks have four input variables: position of cart x, angle of pole θ_1, velocity of cart $\dot{x}$ and the velocity of pole $\dot{\theta_1}$. So the neurons in the FCGPANN network for this case each have four inputs [8] while in the RCGPANN network it was set to five. The fifth input to the nodes in the first column is supplied with a connection from the state unit. The layers following that might or might not have a connection from the state unit.

In the non-Markovian case, the network has only two inputs: position of cart x and the angle of pole θ_1. So the number of node inputs (arity) to each FCGPANN neuron was set at 2, while for the RCGPANN it was set at 3, where, as usual, *initially* the third input in the first column of neurons is always a connection to the state unit. As the velocity information is not provided to the network, the controller has to internally compute the velocities. The feedback connections are important to make this possible [36,10]. As the velocity is obtained through differentiation of other parameters, and a differential equation cannot be solved without a feedback connection, this is why feedforward networks are unable to solve the problem when velocity information is not provided.

Tables 3–5 represent the comparison of the average number of balancing attempts (evaluations) required to achieve a solution under various conditions. Results are shown for cases where network inputs are initialized to zero or with random values within the range (as mentioned in Section 5). Various network sizes and various mutation rates were also investigated. In these tables we tabulate the average number of genotype evaluations over 50 runs and also the coefficient of variation (CV). The CV is the ratio of the standard deviation to the mean. It is a useful statistic to calculate as it means that the variation in the results can be compared without reference to the mean.

5.3.2. Double poles: Markovian and Non-Markovian cases

The double pole balancing involves balancing two poles rather than a single pole thus making the problem non-linear. For the Markovian case, the input parameters to the network are position of cart x, angle of both the poles $\theta_{1,2}$, velocity of cart $\dot{x}$ and the velocity of both the poles $\dot{\theta}_{1,2}$. The input to each neuron is set to six for the FCGPANN neuron [8] while it is taken as seven for the RCGPANN neuron. As mentioned in Section 5.3.1, initially each of the neuron(s) in the first column has an input connected to the

Table 3
Performance of FCGPANN on single pole.

Initial state	Network arch.	Mutation rate (%)	Av. No. Evaluations in 50 runs (coefficient of variation)
Zero	5 × 5	10	65(1.47)
	10 × 10	10	41(1.65)
	15 × 15	10	**21(1.93)**
	5 × 5	20	35.48(1.08)
	10 × 10	20	31.4(1.26)
	15 × 15	20	25.32(0.98)
Random	5 × 5	10	477(1.57)
	10 × 10	10	157(1.01)
	15 × 15	10	113(0.81)
	5 × 5	20	172(0.82)
	10 × 10	20	97(0.86)
	15 × 15	20	**95.56(0.66)**

Table 4
Performance of RCGPANN on Single Pole—Markovian Case. Columns 4–6 show average number of evaluations (50 runs). Figures in brackets are coefficient of variation.

Network representation			RCGPANN				Average of (CV)
Initial state	Network arch.	Mutation rate (%)	*S′W′*	*SW*	*S′W*	*SW′*	
Zero	5 × 5	10	42 (1.10)	43 (1.03)	35 (0.82)	44 (0.85)	0.94
	10 × 10	10	**30 (0.87)**	31 (0.62)	28 (0.74)	**26 (0.72)**	0.73
	15 × 15	10	34 (0.85)	28 (1.26)	**17 (0.54)**	28 (0.84)	0.87
	5 × 5	20	45 (0.82)	33 (1.04)	33 (0.87)	38 (0.81)	0.88
	10 × 10	20	35 (0.68)	32 (0.69)	27 (0.77)	29 (0.82)	0.93
	15 × 15	20	30 (0.77)	**23 (1.54)**	23 (0.83)	28 (0.74)	0.76
Random	5 × 5	10	98 (1.20)	86 (1.15)	71 (1.05)	86 (0.9)	1.07
	10 × 10	10	72 (1.04)	70 (0.80)	66 (1.21)	81 (1.44)	1.12
	15 × 15	10	70 (0.85)	**49 (0.79)**	**52 (0.98)**	**51 (1.37)**	0.99
	5 × 5	20	145 (1.93)	65 (0.84)	67 (0.95)	75 (1.30)	1.25
	10 × 10	20	**57 (1.07)**	69 (1.16)	57 (0.88)	77 (0.92)	1.01
	15 × 15	20	86 (0.98)	58 (0.91)	53 (1.03)	50 (0.88)	0.95

Table 5
Performance of RCGPANN on single pole—Non-Markovian Case. Columns 4–6 show average number of evaluations (50 runs). Figures in brackets are coefficient of variation.

Network representation			RCGPANN				Average of (CV)
Initial state	Network arch.	Mutation rate (%)	*S′W′*	*SW*	*S′W*	*SW′*	
Zero	5 × 5	10	299 (2.90)	155 (2.16)	185 (1.90)	197 (1.89)	2.2
	10 × 10	10	106 (1.37)	81 (1.86)	**65 (1.05)**	**61 (0.86)**	1.2
	15 × 15	10	**55 (0.83)**	87 (1.42)	169 (3.90)	71 (0.74)	1.7
	5 × 5	20	73 (1.66)	88 (0.79)	253 (3.11)	96 (0.91)	1.6
	10 × 10	20	78 (1.22)	**57 (0.88)**	85 (1.24)	75 (1.06)	1.1
	15 × 15	20	56 (0.93)	73 (0.81)	89 (1.01)	109 (1.25)	1.0
Random	5 × 5	10	1793 (4)	2541 (3.98)	781 (1.82)	528 (2.03)	2.9
	10 × 10	10	400 (1.66)	497 (1.94)	369 (1.59)	309 (1.03)	1.5
	15 × 15	10	269 (1.70)	**241 (1.22)**	477 (1.25)	505 (1.88)	1.5
	5 × 5	20	723 (4.18)	449 (1.59)	717 (3.45)	**421 (1.60)**	2.7
	10 × 10	20	**215 (1.69)**	343 (1.23)	**281 (1.04)**	351 (1.32)	1.3
	15 × 15	20	238 (1.39)	335 (1.27)	310 (1.27)	363 (1.39)	1.3

state unit. Note that the initial values of input to the state unit is always taken as zero.

For the non-Markovian case, the input to the network is the position of cart x and the angle of poles $\theta_{1,2}$. The input to each FCGPANN neuron is set to three and an RCGPANN neuron is set to four. Since the velocity information is not provided to the controller, networks require feedback to compute the hidden velocity information of the pole(s) and the cart [36,10].

The FCGPANN and the RCGPANN networks are trained independently on the following two fitness functions: standard fitness and the damping fitness function. In both single and double pole balancing tasks, the standard fitness function is used to train the FCGPANN and the RCGPANN networks under Markovian and non-Markovian cases. While a damping fitness function is used on the double poles non-Markovian case only.

The standard fitness function is the number of time steps the network has kept the pole(s) balanced. The damping fitness function devised by Gruau [34] uses two calculations based on behaviour over 1000 time steps expressed by Eqs. (14) and (15). Where all cart variables in Eq. (15) are normalized and f_2 is always capped at 1. The weighted sum in Eq. (16) is taken to compute the overall fitness:

$$f_1 = \frac{t}{1000} \tag{14}$$

$$f_2 = \begin{cases} 0 & t < 100 \\ \dfrac{0.75}{\sum_{i=t-100}^{t}(|x^i| + |\dot{x}^i| + |\theta_2^i| + |\dot{\theta}_2^i|)} & t \geq 100 \end{cases} \tag{15}$$

$$f_3 = 0.1f_1 + 0.9f_2 \tag{16}$$

The intention of introducing the damping fitness function is to force the network to generate internal velocity information and also to penalize oscillations of the pole(s) [36,10].

The evolutionary algorithm used the 1+4-Evolutionary Strategy as defined in procedure 1. We examined two evolutionary scenarios using either the standard fitness function or the damping fitness, in the latter case, a combination of both fitness functions is used.

If a genotype, G_n, is obtained that balances the pole(s) using the damping fitness for 1000 time steps then G_n is examined over 100 000 time steps and the number of steps it can balance the pole (s) is calculated. If it balances the pole(s) for 100 000 time steps the algorithm stops as it has been successful, if however, G_n cannot balance the pole for 100 000 time steps then the *parent* of G_n is mutated four times, each mutated parent becomes a member of the population. This together with the parent forms the population at the next generation. Tables 6–9 represent the average number of evaluations to successfully balance poles (for 100 000 time steps) of the evolved genotypes in fifty (50) independent runs, for initial states of zero and random values and for both the feedforward [8] and recurrent networks with varying network sizes and mutation rates. Tables 8 and 9 show the evaluations for the standard and damping fitness functions, where the minimum average number of evaluations for each experimental scenario is marked in bold.

5.4. Results and discussion

5.4.1. Single pole: Analysis

Table 3 represents the performance of FCGPANN for the single pole balancing task at Markovian and non-Markovian cases. In the Markovian case, the FCGPANN network produces decreasing number of evaluations as the network size increases. For a 20% mutation rate the 15×15 FCGPANN network was able to find one solution in an average of 21 evaluations (5 generations!). While for the non-Markovian case, the FCGPANN networks were not able to come up with a solution. It is also observed that networks initialized with random values take longer (time) to get balanced as compared to those initialized with zero initial states.

Table 4 shows the performance of the RCGPANN on the single pole problem using different network parameters when complete information is provided to the network. Once again solutions with random initialization of cart-pole variables take longer to evolve than with zero initialization. Also it is observed that the average coefficient of variation of a 5×5 network is higher than the other networks. This means that 5×5 network performs poorer as it takes a longer time to produce solutions. With increasing network size the average evaluations of all the RCGPANN networks exhibited random behaviour. Out of the four possible ways of training a recurrent network, a minimum average evaluation of 17 was achieved for a 15×15 network with 10% mutation rate. A mutation strategy (*SW′*) that involves mutating the weight of the state units

Table 6
Performance of FCGPANN on double pole.

Initial state	Network arch.	Mutation rate (%)	Average number of evaluations over 50 runs (coefficient of variation)
Zero	5×5	10	205(1.26)
	10×10	10	93(0.85)
	15×15	10	**77(2.27)**
	5×5	20	141(1.28)
	10×10	20	97(1.96)
	15×15	20	93(0.96)
Random	5×5	10	1183(3.54)
	10×10	10	437(2.69)
	15×15	10	181(1.75)
	5×5	20	297(1.21)
	10×10	20	205(1.08)
	15×15	20	**162(1.01)**

Table 7
Performance of RCGPANN on double pole—Markovian case. Columns 4–6 show average number of evaluations in 50 runs (coefficient of variation).

Network representation			RCGPANN				Average of (CV)
Initial state	Network arch.	Mutation rate (%)	*S′W′*	*SW*	*S′W*	*SW′*	
Zero	5×5	10	229 (1.14)	181 (0.96)	**290 (1.03)**	342 (1.18)	1.8
	10×10	10	213 (1.11)	**129 (1.17)**	329 (1.14)	417 (1.76)	1.3
	15×15	10	**197 (0.90)**	178 (1.60)	349 (1.34)	**255 (0.93)**	1.1
	5×5	20	309 (1.12)	239 (0.72)	333 (0.81)	453 (1.56)	1.4
	10×10	20	289 (0.82)	239 (0.83)	354 (0.84)	649 (2.14)	0.95
	15×15	20	224 (1.05)	173 (0.88)	661 (1.37)	541 (1.20)	1.1
Random	5×5	10	419 (1.62)	499 (2.92)	569 (1.67)	546 (1.16)	1.6
	10×10	10	335 (1.17)	264 (1.62)	**389 (1.01)**	365 (1.2)	1.7
	15×15	10	**241 (1.03)**	**217 (1.16)**	1065 (1.74)	277 (1.29)	1.3
	5×5	20	314 (1.05)	436 (1.91)	1013 (1.85)	697 (1.71)	1.1
	10×10	20	941 (2.61)	331 (1.14)	1153 (2.5)	1357 (1.50)	1.8
	15×15	20	569 (1.41)	394 (2.09)	933 (1.32)	1393 (2.17)	1.5

Table 8
Performance of RCGPANN on double pole: non-Markovian case using standard fitness function. Columns 4–6 show average number of evaluations in 50 runs (coefficient of variation).

Network representation			Standard fitness				Average of (CV)
Initial state	Network arch.	Mutation rate (%)	*S′W′*	*SW*	*S′W*	*SW′*	
Zero	5 × 5	10	294 (0.84)	421 (2.49)	705 (3.35)	1481 (3.08)	1.7
	10 × 10	10	**216 (0.88)**	**163 (0.92)**	**249 (0.84)**	285 (1.58)	1.7
	15 × 15	10	231 (1.56)	365 (2.22)	681 (3.22)	**257 (0.84)**	2.2
	5 × 5	20	469 (1.15)	461 (0.86)	401 (1.46)	733 (1.89)	1.4
	10 × 10	20	317 (0.69)	712 (2.06)	493 (1.18)	549 (1.13)	1.4
	15 × 15	20	409 (0.97)	689 (1.08)	733 (1.04)	697 (1.77)	1.5
Random	5 × 5	10	1033 (2.11)	1393 (2.59)	3243 (2.16)	3639 (2.57)	2.1
	10 × 10	10	1215 (3.04)	**674 (2.14)**	1841 (2.97)	2397 (3.54)	2.8
	15 × 15	10	**881 (3.78)**	969 (1.88)	**817 (1.12)**	4253 (2.86)	2.6
	5 × 5	20	1789 (1.80)	2061 (2.44)	1853 (1.58)	**1489 (1.57)**	2.5
	10 × 10	20	1725 (3.03)	2081 (2.38)	3361 (2.14)	2229 (1.43)	2.3
	15 × 15	20	1302 (2.15)	3125 (2.52)	2077 (1.93)	1625 (1.60)	1.9

Table 9
Performance of RCGPANN on double pole: non-Markovian using damping fitness function. Columns 4–6 show average number of evaluations in 50 runs (coefficient of variation).

Network representation			Damping fitness				Average of (CV)
Initial state	Network arch.	Mutation rate (%)	*S′W′*	*SW*	*S′W*	*SW′*	
Zero	5 × 5	10	2308 (3.93)	881 (2.89)	809 (2.53)	1353 (2.51)	2.6
	10 × 10	10	1074 (1.91)	1049 (3.29)	1023 (4.44)	**621 (3.29)**	2.2
	15 × 15	10	429 (1.16)	1401 (4.68)	**394 (1.16)**	901 (3.46)	2.3
	5 × 5	20	917 (2.2)	981 (2.13)	529 (1.27)	801 (0.98)	3.6
	10 × 10	20	**387 (0.83)**	889 (1.31)	1117 (2.05)	1677 (3.81)	1.8
	15 × 15	20	450 **697** (1.16)	**(1.04)**	929 (1.15)	1357 (1.33)	1.1
Random	5 × 5	10	2349 (2.95)	3073 (3.56)	1701 (1.91)	1929 (2.18)	2.9
	10 × 10	10	2184 (2.34)	2181 (3.04)	1695 (2.57)	**1569 (2.88)**	2.9
	15 × 15	10	1581 (2.07)	**949 (1.73)**	**1692 (3.41)**	3429 (3.10)	2.3
	5 × 5	20	4149 (2.52)	2181 (3.80)	4089 (2.25)	2185 (2.32)	2.4
	10 × 10	20	**1007 (1.19)**	3361 (3.13)	3857 (3.16)	1901 (1.12)	2.8
	15 × 15	20	1409 (2.41)	2121 (2.47)	1705 (1.47)	3765 (2.53)	2.5

while keeping the switches always enabled (1) gave the best result.

Table 5 shows the RCGPANN results for the non-Markovian case with various network sizes and mutation rates.

Out of the four different types of evolutionary mechanisms the 15×15 structure with a mutation rate of 10% produced the minimum average number of evaluations (55). This is the mechanism (*S′W′*) where both the weight and switches of the state unit are not mutated. This forces the algorithm to produce recurrent solutions only. Since during evolution the mutation of the weights of state units and switches is disabled, feedback thus cannot be removed. So the networks produced are always recurrent. This shows that these parameters do not affect the ultimate solutions to a large extent. Similar to Table 4, the average coefficient of variation for a 5×5 network is higher than the rest of the network sizes. Thus a 5×5 network requires more evaluations on average to find a solution than the larger networks. Also it is observed that the average evaluations for a Markovian case (Table 4) are less than the non-Markovian case (Table 5). Thus, the solutions in the non-Markovian case is clearly more difficult to evolve.

5.4.2. Double poles: Analysis

Table 6 demonstrates the performance of FCGPANN on the double pole for both Markovian and non-Markovian cases. Under Markovian case, a similar trend in average number of evaluation was observed as seen in Table 3. Using a mutation rate of 10% with a network size of 15×15 the FCGPANN network was able to find a solution in an average of 77 evaluations. While for the non-Markovian case FCGPANN was unable to find any solution. Similar behaviour was also observed in the single pole balancing task as shown in Table 3. Thus for non-Markovian case feedback connections are required to generate solutions [9].

Fig. 7(a,c) displays the neural network structures of the FCGPANN and RCGPANN evolved genotypes for the Markovian and Non-Markovian cases with random initial values respectively. Fig. 7(b,d) represents the corresponding pole angles and position of cart simulated for 30 min (100 000 steps are down-sampled to produce 100 steps for demonstration purpose only), whereas in Fig. 7(a,c) SU represents the state unit. The State Unit is mathematically expressed as shown in

$$SU(t) = \tanh(SU(t-1) + F(t-1)) \tag{17}$$

Eqs. (18) and (19) represent the mathematical expressions for the network in Fig. 7(a,c):

$$F(t) = \tanh(-0.62\dot{x} + 0.5\dot{\theta_1} + 0.309\theta_2 + 0.05\dot{\theta_2}) \tag{18}$$

$$F(t) = \tanh(-0.336x + 0.77\theta_1 + \theta_2 - 0.87SU(t)) \tag{19}$$

Table 7 shows the average evaluations of the RCGPANN networks on the double pole for the Markovian case. Here no clear trend was observed in the average evaluations when network structure and mutation rate were varied. Examining the results for the four different state unit mutation strategies, we observe that the best results are obtained for *CW*, where state unit switches and weight are allowed to be mutated. This scenario produced the best possible results by solving the task in minimum number of evaluation on an average. An average of 129 evaluations was required to evolve successful double pole balancing using a 10×10 network structure with a mutation rate of 10%.

Tables 8 and 9 show the average number of evaluations for the RCGPANN networks for the double pole non-Markovian cases using the standard and the damping fitness functions. It is observed that the RCGPANN network finds solution easily using the standard fitness as compared to the damping fitness function since the average number of evaluations in Table 8 is smaller than in Table 9.

With increasing network size and constant mutation rates the number of average evaluations shows an apparently random trend. However smaller networks like 5×5 produced consistently poorer results by exhibiting high value of average of coefficient of

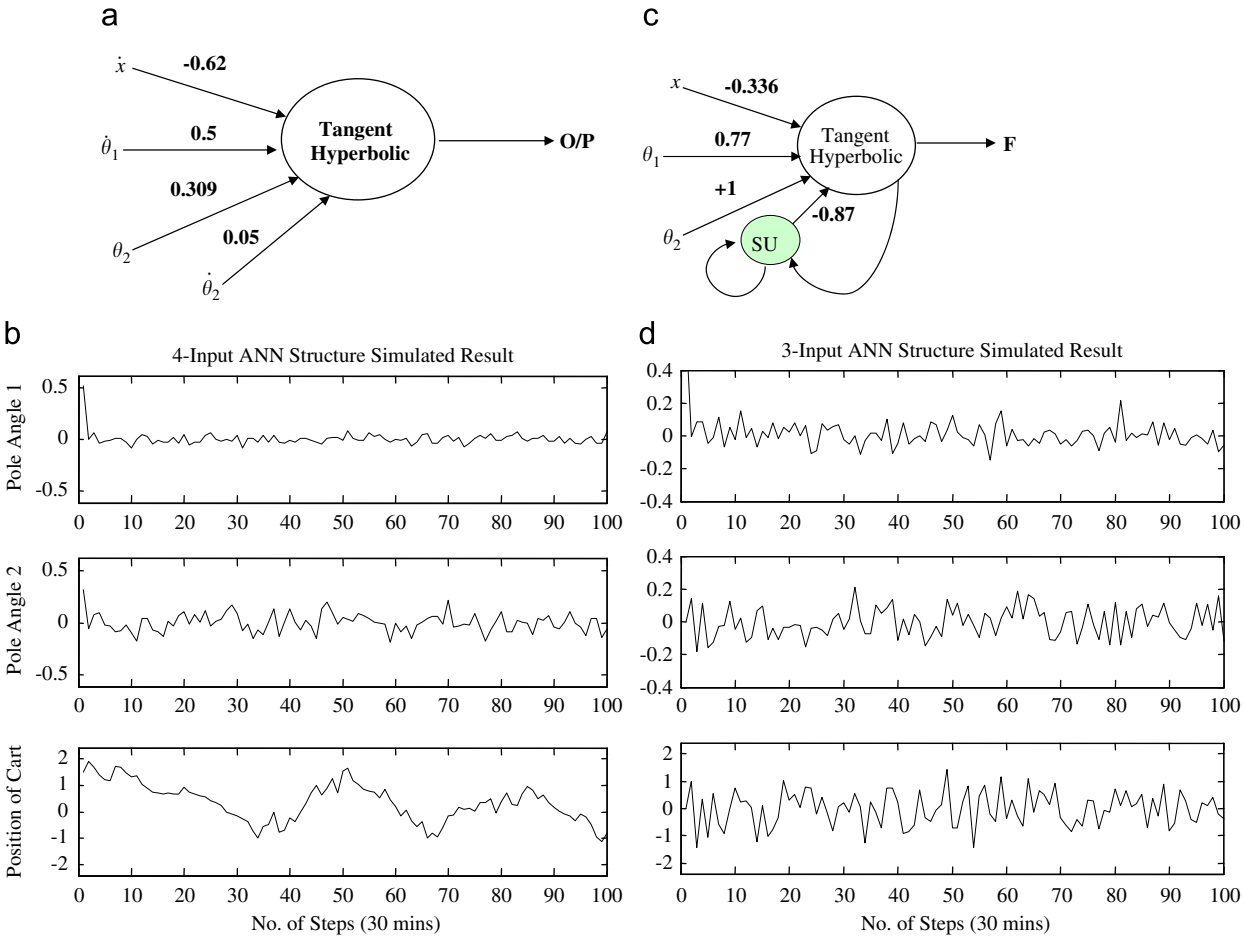

Fig. 7. FCGPANN: Double pole balancing task-Markovian case (a) phenotype of the evolved genotype at random initial state, (b) pole angle and position of cart simulated for the ANN in (a) for 100 000 steps, (c) phenotype of the evolved genotype without velocity, (d) pole angle and position of cart simulated for the ANN in (c) for 100 000 steps.

Table 10
Comparison of CGPANN with other neuroevolutionary algorithms applied on single pole balancing task: average number of network evaluations.

Method	Markovian	Non-Markovian
CNE	352	724
SANE	302	1212
ESP	289	589
NEAT	743	1523
CoSyNE	98	127
FCGPANN	**21**	–
RCGPANN	**17**	**55**

variation as evident in Table 9. From the four different methods of evolving the state unit parameters, a network size of 10×10 with mutation rate 10% was able to find the solution at an average of 163 evaluations for the standard fitness function, while at 20% mutation rate the 10×10 balanced the poles within 387 evaluations for the damping fitness function.

Table 10 represents the comparison of feedforward and recurrent CGPANN with other neuroevolutionary techniques for Markovian and non-Markovian cases explored on single pole balancing task. The FCGPANN network has generated solutions at an average of 21 evaluations and the RCGPANN at an average of 17 evaluations for the Markovian case. FCGPANN was unable to find any solution at non-Markovian case while RCGPANN produced solutions at an average of 55 evaluations. The proposed algorithms clearly outperformed and produce solutions in fewer evaluations, on average, than all other neuroevolutionary techniques proposed to date.

Table 11 represents the comparison of feedforward and recurrent CGPANN with other neuroevolutionary techniques for both Markovian and non-Markovian cases for the double pole balancing task. For the Markovian case the FCGPANN found a solution in an average of 77 evaluations (15×15 network with mutation = 10% see Table 6) , while the RCGPANN found a solution in a minimum of 129 (on an average) evaluations (in 50 runs). The FCGPANN was unable to produce solution for the non-Markovian case as the network does not allow feedback connections which is essential for the computation of velocities [36,10]. For the RCGPANN networks the minimum number of average evaluations over 50 runs was 163 for the standard fitness and 387 for the damping fitness function. Once again we observe that the average number of evaluations required to evolve a solution is considerably fewer than all previously published figures of other neuroevolutionary algorithms.

It is important to mention that the performance of both the FCGPANN and RCGPANN is based on a number of parameters namely mutation rate, network size, number of inputs to each node, averaging the number of outputs and the functions used. Thus a more thorough investigation of these parameters are likely to produce even better results.

5.5. Generalization

In addition to the learning speed of evolution, the robustness of the proposed neuroevolutionary algorithm was tested. We examined

Table 11
Comparison of CGPANN with other neuro-evolutionary algorithms in terms of average number of evaluation required to solve the double pole balancing task.

	Markovian	Non-Markovian	
	Standard fitness	Standard fitness	Damping fitness
CNE	22 100	76 906	87 623
SANE	12 600	262 700	451 612
ESP	3800	7374	26 342
NEAT	3600	–	6929
CoSyNE	954	1294	3416
FCGPANN	**77**	–	–
RCGPANN	**129**	**163**	**387**

Table 12
Generalization of CGPANN: average number of random cart initializations (out of 625) that can be solved.

Algorithm type	Markovian		Non-Markovian	
	Single Pole	Double Pole	Single Pole	Double Pole
FCGPANN	590	277.38	–	–
RCGPANN	363.94	471.92	294	335.84

Table 13
Comparison of generalization of CGPANN with other neuroevolutionary algorithms.

Method	Value
CE	300
ESP	289
NEAT	286
RCGPANN	**335.84**

the ability of evolved solutions to generalize in other cases where network inputs are randomly initialized. We created 625 such random initial cart and pole(s) states and looked at the manner by which many evolved networks successfully balanced the pole(s) over 1000 steps.

Table 12 displays the generalization figures of CGPANN under Markovian and non-Markovian conditions for both single and double pole balancing tasks. A total of 50 independent evolved genotypes are tested on 625 random initial states for generalization. It was observed that for the single pole Markovian case RCGPANN scored 363.94 while FCGPANN scored 590. This indicates that the evolved networks are robust solutions and also confirms that linear problems solved by feedforward networks are more robust. Using recurrent connections for linear problem adds noise to the network. This ultimately affects the networks capability to converge and to produce generalized networks.

For the double pole Markovian case even though the FCGPANN approach had performed faster in evolving the solution (as shown in Table 11) the RCGPANN were found more robust with a value of 417.92 as compared to 277.38. This indicates that non-linear problems are effectively and efficiently handled by recurrent networks and they are able to produce general and robust solutions.

Table 13 presents the generalization ability of various neuroevolutionary algorithms for the double pole balancing scenario for non-Markovian case using the damping fitness function. It is observed that the RCGPANN scored 335.84 out of 625 for 50 independent evolved genotypes exhibiting greater generalization ability as compared to other techniques. This shows that the solutions (evolved genotypes) can cope with a large number of different initial conditions and that many solutions exhibit general behaviour.

6. Case study-II: Application of CGPANN to breast cancer diagnosis

Breast cancer is one of the leading causes of death in women. Detection of the disease at an earlier stage can save precious lives. Different diagnostic tests and procedures are available for it. One of these is the biopsy of the breast, which is quite painful and causes discomfort to the patient but is more reliable than others. Biopsy however is needed in order to ascertain whether a tumor is benign or malignant. Due to the discomfort associated with the biopsy, patients often hesitate to visit a doctor until it is too late. A less invasive test is mammography, in which the radiologist examines the X-rays of the breast for a possible mass. However accuracy of this test largely depends on the expertise of the radiologist and consequently there is a chance that a malignant lesion is diagnosed as benign or vice versa. Thus it is desirable to find a low invasive yet reliable test. One such method is the application of computational intelligence to the data obtained using Fine Needle Aspiration (FNA) [57].

There are a number of papers reporting the application of computational intelligence in breast cancer detection [58] including the multilayer perceptron [59], radial basis function (RBF) networks [60], fuzzy classifiers [61,62], clustering algorithms [63], evolutionary computation [64], principal component analysis [62], and different kernel-based methods [65].

The work we are presenting in this paper is based on classification of two data sets. The first of these is the original FNA data set [57] while the second one available at WDBC web site[1] is the processed form of the first set. Cartesian Genetic Programming evolved Artificial Neural Networks (CGPANNs) [66] are used in separate experiments to classify each of these data sets. The system takes as input, the FNA data and classifies the case as either benign or malignant. Before the network can be applied for diagnosis it must be trained first. The training process consists of applying to the network, a part of the data from one of the data sets, that includes the FNA feature parameters and the corresponding classification results (either benign or malignant). Once trained, the network is capable of diagnosing using unseen FNA data (not used in training) whether a breast mass is benign or malignant.

6.1. Previous work

Different methods have been explored to diagnose the breast cancer. In [67] Genetic Programming (GP) is used for feature generation using Fischer criterion from a raw data set in a pattern recognition problem. The features thus extracted are fed to an ANN for classification.

Fisher Linear Discriminant Analysis (FLDA) and principal component analysis (PCA) were previously considered the best feature extraction methods [68].

In another approach, as discussed in [69] the preprocessed data from WDBC is fed directly as terminal values to Genetic Programming Algorithm. The output of this is compared with the desired output and the fitness is computed. A population of individuals is generated and the best individuals are selected on the basis of fitness. Genetic operations of crossover and mutation are applied to these individuals. Following a number of generations, the

[1] Breast Cancer Wisconsin (Diagnostic) Data Set http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

population converges to the solution that best represents the discrimination function.

In [70] a limited database of mammograms was used to shape features of the breast masses that were classified by genetic programming as either benign or malignant. To refine features applied to the GP classifier, feature-selection methods including student's *t* test, Kolmogorov–Smirnov test, and Kullback–Leibler divergence were used. Land et al. [71] used a modified form of Fogel's evolutionary programming for evolution of NN for breast cancer detection. With the same FNA data set that we used they could achieve an average success % age of about 97.26%. Mu et al. discuss the application support vector machines (SVMs), radial basis function (RBF) networks, and self-organizing maps (SOMs) for breast cancer detection [72] obtaining competitive results.

In SOM-RBF classifier the RBF network processes the clustering result obtained by the SOM. Experimental results obtained with the proposed method show a detection accuracy of up to 98%.

Werner et al. in their paper address the problem of how to obtain a mathematical discrimination function for quantifying the severity of a disease using genetic programming (GP) [69]. A detection accuracy of around 96.32% was attained using their approach. They compared their result of GP with those of West et al. [73]. They have explored the results for diagnosis of breast cancer using MLP, general regression (GR), RBF, mixture of experts (MOE), and other methods.

Hussein et al. [74] discuss an evolutionary artificial neural network (EANN) approach based on pareto-differential evolution (PDE) algorithm with local search for breast cancer detection. The approach is named memetic Pareto artificial neural network (MPANN). An accuracy of 98.1% was obtained using MPANN.

Majid et al. [58] discussed the benefits of applying support vector machines (SVMs) and radial basis functions for breast cancer detection [58]. They compared their results using the method of L2-SVM/GDVEE(RBF): 98.1% accuracy to those of linear SVM classifier [67]: 94% , the fuzzy classifier [75]: 95.8%, and the edited nearest-neighbor (ENN) with pure filtering [76]: 95.6%.

Janghel et al. [77] had compared various neural network models used for the breast cancer diagnosis. They implemented six models of neural networks namely: back propagation algorithm, radial basis function networks, learning vector quantization, probabilistic neural networks, recurrent neural networks, and competitive neural networks.

Beatriz et al. [78] discuss the Artificial Bee Colony (ABC) algorithm for optimization of Artificial Neural Network (ANN) resulting in improved performance and reduced size of the network. Without optimization there is a likelihood that ANN is trapped in local minima. The synaptic weights, architecture and transfer function are evolved and also minimum classification error (CER) and minimum Mean Square Error (MSE) are obtained. The algorithm was evaluated besides others with the Breast cancer data set taken from UCI repository. Beatriz et al. [79] also implemented the Particle Swarm Optimization (PSO) algorithm for evolving the synaptic weights, the topology and the transfer functions of each neuron of an ANN and tested the technique on a number of nonlinear problems including the breast cancer.

6.2. Diagnostic procedures and data set

The procedure for this test consists of extracting the mass from the suspected region of the breast cancer patient using Fine Needle Aspiration (FNA). The mass extracted is examined under microscope for the following nine features: (1) clump thickness, (2 and 3) uniformity of cell size and shape, (4) marginal adhesion, (5) single epithelial cell size, (6) bare nuclei, (7) bland chromatin, (8) normal nucleoli and (9) mitoses [57].

Based on these features the physician decides whether a sample is benign (non cancerous) or malignant (cancerous), each of these features has a value between 1 and 10, however no single feature can be made a basis for decision. The set consists of 699 instances, out of which 458 are benign and 241 are malignant. A data set of patients based on these parameters was collected by the physician W.O. Wolberg from patients at University of Wisconsin hospitals.

However, for machine learning another data set is formed by processing the digital images of the well differentiated cells of the masses with a computer, for the following features: (1) Radius, (2) Perimeter, (3) Area, (4) Compactness, (5) Smoothness, (6) Concavity, (7) Concave points, (8) Symmetry, (9) Fractal dimension and (10) Texture [72]. The mean value, worst (mean of the three largest values) and standard error of each feature are computed for each of these parameters and as such there are a total of 30 feature values [80]. A patient data set with these feature values and their results is available at the WDBC web site. The set consists of 569 patterns, out of which 357 are benign and 212 are malignant.

We evaluated our network with data from both of these data sets. For the first set of experiments the input to the network consisted of nine features while for the second there were 30 features. In each case there was only one output that has a value of 0 for benign and 1 for malignant. To obtain this, we round to the nearest integer the real-value output of our network.

6.3. Experimental setup

In these experiments we used a population size of ten. The evolutionary algorithm begins with the generation of ten random genotypes. Each genotype is transformed into phenotype (encoded network), the input data from the training data set is applied and the phenotype evaluated for its fitness by comparing it with target data. Error for each genotype is calculated and the best genotype (having the least error) is promoted to the next generation. The parent genotype and randomly mutated copies of it make the next population. The mutation rate is set at 10% for all the experiments. The number of rows is set to one, meaning that the number of columns is equal to the number of nodes. The process continues from generation to generation until the error or the number of generations reduces to its corresponding pre-set value.

6.4. Error calculation

We have allocated ten outputs from the CGPANN. The outputs are averaged and compared to a threshold of 0.5 (output of Neuron is in the range of 0 and 1), if the averaged output is equal to or greater than 0.5 the CGPANN classifies the sample as malignant (1), otherwise it classifies the sample as benign (0). All the predicted outputs are then compared with the actual target, and the total error is calculated along with Type-I and Type-II errors (see in next paragraph). From this the percentage error is calculated for each category. This is the indication of how accurate the system prediction is.

There are two types of errors, Type-I and Type-II. In Type-I error a benign sample is falsely classified as malignant, while in Type-II error a malignant sample is falsely classified as benign. The second type is a more catastrophic mistake. The sum of errors resulting from the application of complete training data to a genotype gives the fitness of that genotype in terms of the total error. All the genotypes of the population are evaluated in this way and passed on to the next generation. We calculate the total error, error for type-I and type-II, using the following formulae:

$$\text{Total Percentage Error} = \text{Fitness of the genotype (total number of false predictions)} \times 100/N_t$$

$$\text{Type}-\text{I Percentage Error} = N_{T-I} \times 100/N_t$$

$$\text{Type}-\text{II Percentage Error} = N_{T-II} \times 100/N_t$$

Table 14
Table of results for various maximum number of nodes, inputs per node (arity) for a single run. The training data set used data for 200 patients and the test data set used data from 200 different patients.

Number of nodes	Number of inputs per node	Training accuracy (%age)	Testing accuracy (%age)	Testing Type-1 accuracy (%age)	Testing Type-2 accuracy (%age)
100	5	98.5	95.5	97.5	98
	10	97	95	97.5	97.5
	15	98.5	95.5	98	97.5
	20	98.5	96	98	98
	25	98.5	94.5	95.5	99
	30	98.5	96	97.5	98.5
	35	99	95	95.5	**99.5**
	40	98.5	94	96.5	97.5
200	5	98	97	98.5	98.5
	10	97.5	95.5	97.5	98
	15	97	92.5	93.5	99
	20	99	96	97.5	98.5
	25	98	95	96	99
	30	97.5	95	96.5	98.5
	35	**99.5**	96.5	97	**99.5**
	40	**99.5**	97	98	99
300	5	98	95	95.5	**99.5**
	10	98	96	97.5	98.5
	15	99	94.5	96.5	98
	20	98.5	95	97	98
	25	**99.5**	**98**	**99**	99
	30	96	95	96	99
	35	**99.5**	95.5	96.5	99
	40	99	95	96	99

whereas N_t=Total number of samples diagnosed, N_{T-I} = Number of benign patients diagnosed as malignant, N_{T-II} = Number of malignant patients diagnosed as benign.

6.5. Results and analysis

In order to evaluate the algorithm, initially we ran 24 experiments with different morphologies of the network in terms of the maximum number of nodes and number of inputs per node. The best result obtained so far has one error while training with 200 cases i.e. 99.5% successful training. We ran all the experiments for one hundred thousand generations. Once the training was complete, we tested the evolved model on another set of 200 test cases and the best results obtained gave 4 errors out of 200 cases thus giving 98% accuracy as a whole. Table 14 shows the results obtained for all the cases with number of nodes 100, 200 and 300 and the corresponding number of inputs per node varying from 5 to 40 with a step size of 5. The data for both training and testing is selected randomly from the complete set for all the experiments.

Table 14 demonstrates a promising trend in the results with an average accuracy of 96%. It is encouraging that in these experiments most of the errors are of Type-I (99%, two errors out of two hundred samples) which falsely classify benign sample to be malignant, while the maximum number of Type-II errors is only 1 (99.5% accurate). This means that there is very little probability that a patient having malignancy would be classified as healthy which would be a serious error in comparison to the case that a healthy person be classified as having cancer. The latter would merely indicate that more tests were required.

Table 17 compares the results of all previously published methods using the WDBC data set with 30 features, with the results obtained through CGPANN using the same data set, while Table 18 compares the results of all previous published methods using the original data set that has nine features with the results from CGPANN using the same data set. This clearly demonstrates the accuracy of CGPANN.

7. Further analysis

In order to evaluate and compare CGPANN with other algorithms we arranged the experimental setup such that it complied with the previously published work. In order to do this, we arranged the data for ten (10) fold cross validation. In this method each data set is divided into 10 blocks of approximately equal size. The data is then shuffled to create ten different data sets. At the end the network is trained with the 9 blocks and tested with the tenth (10th) block. After arranging the data, we selected the best optimal network as highlighted in Table 14 with 200 nodes and 35 inputs per node as the base parameters.

The 10 fold cross validation was performed with the data sets containing 30 features as well as that containing nine features. In the first case the nine block data set represent 512 patients data samples with each sample having 30 feature inputs and one output. The network was trained with this data and then tested with the remaining 57 samples. During training we adjusted various network parameters (weights and topology) in hundred thousand generations in order to obtain the best possible model. This process of training and testing was repeated for the other shuffled data sets as well, and their average training and testing accuracy were calculated. The above procedure was repeated ten times, each time with a different seed for the random number generator used for making the initial population. Table 15 shows the result of the experiments run using this data set.

In the second case we applied the same 10 fold cross validation procedure for the FNA data set that contains nine features. In this case the 699 patient samples were divided into 10 data blocks of about 70 each. Table 16 shows the results of the experiments run with this data set.

From these tables it is evident that the network does perform well on an average and attains 100% accuracy at times for both type-I and type-II cases. The most encouraging aspect is that type-II error averages at 98.5% which shows the strength of algorithm in avoiding the misclassification of malignant cases as benign.

8. Conclusion and future work

In this paper a fast learning neuroevolutionary algorithm based on Cartesian genetic programming (CGPANN) of both feedforward (FCGPANN) and recurrent (RCGPANN) architecture was proposed.

Table 15
Cross-validation average results of various sets of patient data (FNA data set with thirty features) over ten independent runs each.

Data set	Training accuracy (%age)	Testing accuracy (%age)	Testing Type-1 accuracy (%age)	Testing Type-2 accuracy (%age)
1	97	93	95	98
2	99	95	96.5	98
3	98	95	98	96.5
4	99	95	98	96.5
5	99	95	96.5	98
6	98	98	**100**	98
7	98	98	98	**100**
8	97	98	98	100
9	98.5	95	96.5	98
10	99	95	95	100
Average	98.5	96	97	98.5

Table 16
Cross-validation average (ten seeds) results of various sets of patient data (FNA data set with nine features) over ten independent runs each (total hundred (100) experiments).

Data set	Training accuracy (%age)	Testing accuracy (%age)	Testing Type-1 accuracy (%age)	Testing Type-2 accuracy (%age)
1	98.41	98.62	98.57	99.84
2	99.21	97.94	99.05	100
3	99.21	97.78	99.21	100
4	99.21	97.78	99.21	100
5	99.29	98.15	99.29	100
6	98.57	97.78	98.57	100
7	99.29	97.94	99.29	100
8	99.29	97.99	99.29	100
9	99.21	98.15	99.21	100
10	98.57	97.78	99.21	99.21
Average	99.02	97.99	99.09	99.90

Table 17
Comparison of Mean Absolute Percentage Errors (MAPE) obtained using various classification methods using the processed FNA data from WDBC that contains 30 features.

No.	Method	Mean (MAPE)	References
1	MLP	95.56	[67]
2	SVM	93.95	–
3	FLDA/MLP	90.92	–
4	PCA/MLP	92.02	–
5	GP/MDC	96.58	–
6	SOM-RBF	98	[72]
7	MLP	95.7206	[69]
8	GR	96.7647	–
9	RBF	97.0441	–
10	MOE	96.2941	–
11	LDA	96.33968	–
12	Logistic	97.2182	–
13	K neighbour	96.7789	–
14	Kernel	95.022	–
15	GP - Test average	96.3235	–
16	L2-SVM/GDVEE (RBF)	98.1	[58]
17	SVM (linear)	94.0	–
18	SVM (RBF)	97.7	–
19	Fuzzy	95.8	–
20	ENN	95.6	–
21	CGPANN	**97 for Type-I and 98.5 for Type-II**	Proposed Solution

Table 18
Comparison of Mean Absolute Percentage Errors (MAPE) obtained using various classification methods using the FNA data with nine features.

No.	Method	Mean (MAPE)	References
1	EPNet	97.254	[71]
2	MPANN	98.1	[74]
3	BP A	98.14	[77]
4	RBF	89.96	–
5	LVQ	46.10	–
6	PNN	97.77	–
7	RNN	97.77	–
8	CL	64.31	–
9	NLCS with GA	97.0	[81]
10	NegBoost with $\lambda = 0$	98.3	[82]
9	CGPANN	**for Type-I and for Type-II**	Proposed Solution

The algorithm was tested on the pole-balancing benchmark and for the diagnosis of breast cancer.

The results presented in this paper demonstrate that CGPANN extends the powerful and flexible representation of CGP to the evolution of neural networks. It was observed that FCGPANN and RCGPANN generated solutions with fewer evaluations on an average for pole balancing tasks as compared to other published techniques. Adding recurrency to the CGPANN network improved the performance of the network by finding solutions to non-linear and non-Markovian states. Simulation results demonstrate that the FCGPANN and RCGPANN produce robust controllers with better generalization ability than other methods reported in the literature to date. Clearly CGPANN is highly competitive with established neuroevolutionary techniques such as NEAT, SANE, ESP and CoSyNE.

We have also observed the fast learning capabilities of CGPANN in the accurate diagnosis of Breast cancer. The work presented shows promising results and can be explored further for mammogram analysis in addition to FNA data classification. This will need extra effort and some preprocessing of the mammograms images before being supplied to the CGPANN classifier.

Future work in CGPANN also involves a detailed analysis by modifying the CGPANN algorithm to incorporate levels back (a connectivity parameter in CGP), the neuron arity, adaptive mutation rate and adaptive network size. Also modular ANNs using Embedded CGP [20] can be investigated. CGPANN will also be applied on a range of AI problems to explore its capability in diverse application domains.

References

[1] P.M. Wong, T. Gedeon, I.J. Taggart, An improved technique in porosity prediction: a neural network approach, IEEE Trans. Geosci. Remote Sensing 33 (4) (1995) 971–980.
[2] Y. Wu, T.S. Huang, Nonstationary color tracking for vision-based human-computer interaction, IEEE Trans. Neural Netw. 13 (4) (2002) 948–960.
[3] S.M. Tetsuji Tani, M. Umano, Neuro-fuzzy hybrid control system of tank level in petroleum plant, IEEE Trans. Fuzzy Syst. 4 (3) (1996) 360–368.
[4] J.A. Farrell, M.M. Polycarpou, Adaptive Approximation Based Control: Unifying Neural, Fuzzy and Traditional Adaptive Approximation Approaches, Wiley-Blackwell, 2006.
[5] K. Stanley, R. Sherony, N. Kohl, R. Miikkulainen, Neuroevolution of an automobile crash warning system, in: Proceedings of GECCO-2005, 2005.
[6] K. Tumer, Coordinating multi-rover systems: evaluation functions for dynamic and noisy environments, in: GECCO-2005, ACM Press, 2005, pp. 591–598.
[7] J. Koutnik, F. Gomez, J. Schmidhuber, Evolving neural networks in compressed weight space, in: GECCO '10, ACM, 2010, pp. 619–626.
[8] M.M. Khan, G.M. Khan, J.F. Miller, Evolution of optimal ANNs for non-linear control problems using cartesian genetic programming, in: Proceedings of IEEE. ICAI, 2010, pp. 339–346.
[9] F. Gomez, J. Schmidhuber, R. Miikkulainen, Accelerated neural evolution through cooperatively coevolved synapses, J. Mach. Learn. Res. 9 (2008) 937–965.
[10] D. Moriarty, Symbiotic Evolution of Neural Networks in Sequential Decision Tasks, Ph.D. Dissertation, University of Texas at Austin, 1997.
[11] N. García-Pedrajas, C. Hervás-Martínez, J. Muñoz-Pérez, Covnet: a cooperative coevolutionary model for evolving artificial neural networks, IEEE Trans. Neural Netw. 14 (3) (2003).
[12] J. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.
[13] W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francone, Genetic Programming—An Introduction; On the Automatic Evolution of Computer Programs and its Applications, Morgan Kaufmann, 1998.
[14] R. Poli, W.B. Langdon, N.F. McPhee, A Field Guide to Genetic Programming, Lulu, 2008.
[15] J.F. Miller, P. Thomson, T.C. Fogarty, Designing electronic circuits using evolutionary algorithms, arithmetic circuits: a case study, in: Genetic Algorithms and Evolution Strategies in Engineering and Computer Sciences, Wiley, 1998, pp. 105–131.
[16] J.F. Miller, An empirical study of the efficiency of learning Boolean functions using a cartesian genetic programming approach, in: Proceedings of GECCO'99, vol. 2, Morgan Kaufmann, 1999, pp. 1135–1142.
[17] J.F. Miller, P. Thomson, Cartesian genetic programming, in: Proceedings of the 3rd European Conference on Genetic Programming, vol. 1802, 2000, pp. 121–132.
[18] S. Harding, J.F. Miller, Evolution of robot controller using cartesian genetic programming, in: Genetic Programming, 8th European Conference, EuroGP2005, vol. 3447, 2005, pp. 62–73.
[19] J.A. Walker, J.F. Miller, Solving real-valued optimisation problems using cartesian genetic programming, in: Proceedings of GECCO'07, 2007, pp. 1724–1730.

[20] J.A. Walker, J. Miller, Automatic acquisition, evolution and re-use of modules in cartesian genetic programming, IEEE Trans. Evol. Comput. 12 (2008) 397–417.
[21] J.A. Walker, K. Völk, S.L. Smith, J.F. Miller, Parallel evolution using multi-chromosome cartesian genetic programming, Genet. Program. Evolvable Mach. 10 (2009) 417–445.
[22] S.L. Harding, J.F. Miller, W. Banzhaf, Developments in cartesian genetic programming: self-modifying cgp, Genet. Program. Evolvable Mach. 11 (2010) 397–439.
[23] J.C. Bongard, R. Pfeifer, Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny, in: Spector et al. (Ed.), 2001, pp. 829–836.
[24] C. Lee, J.-H. Kim, Evolutionary ordered neural network with a linked-list encoding scheme, in: Proceedings of IEEE. CEC, 1996, pp. 665–669.
[25] X. Yao, Y. Liu, Towards designing artificial neural networks by evolution, Appl. Math. Comput. 91 (1) (1996) 83–90.
[26] D. Dasgupta, D. McGregor, Designing application-specific neural networks using the structured genetic algorithm, in: Proceedings of ICCGANN, 1992, pp. 87–96.
[27] D.W. Opitz, J.W. Shavlik, Connectionist theory refinement: genetically searching the space of network topologies, Journal of Artificial Intelligence Research 6 (1997) 177–209.
[28] B. Zhang, H. Muhlenbein, Evolving optimal neural networks using genetic algorithms with occam's razor, Complex Systems 7 (1993) 199–220.
[29] V. Maniezzo, Genetic evolution of the topology and weight distribution of neural networks, IEEE Trans. Neural Netw. 5 (1) (1994) 39–53.
[30] H. Braun, J. Weisbrod, Evolving feedforward neural networks, in: Proceedings of ICANNGA93, Springer-Verlag, 1993.
[31] F. Gruau, Automatic definition of modular neural networks, Adaptive Behav. 3 (1994) 151–183.
[32] G.S. Hornby, J.B. Pollack, Creating high-level components with a generative representation for body-brain evolution, Artif. Life 8 (2002).
[33] M. Mandischer, Artificial Neural Nets and Genetic Algorithms, Springer, 1993.
[34] F. Gruau, D. Whitley, L. Pyeatt, A comparison between cellular encoding and direct encoding for genetic neural networks, in: Proceedings of First Annual Conference on GP, MIT Press, 1996, pp. 81–89.
[35] X. Yao, Evolving artificial neural networks, in: Proceedings of the IEEE, vol. 87, no. 9, 1999, pp. 1423–1447.
[36] F.J. Gomez, R. Miikkulainen, Solving non-Markovian control tasks with neuroevolution, in: Proceedings of International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1999, pp. 1356–1361.
[37] A.P. Wieland, Evolving neural network controllers for unstable systems, in: Proceedings of International Joint Conference on Neural Networks, 1991, pp. 667–673.
[38] K.O. Stanley, R. Miikkulainen, Efficient reinforcement learning through evolving neural network topologies, in: Proceedings of GECCO'02, Morgan Kaufmann, 2002, pp. 567–577.
[39] Y. Tsoy, V. Spitsyn, Using genetic algorithm with adaptive mutation mechanism for neural networks design and training, in: Proceedings of 9th KORUS'05, IEEE, 2005, pp. 709–714.
[40] K. Ohkura, T. Yasuda, Y. Kawamatsu, Y. Matsumura, K. Ueda, Mbeann: mutation-based evolving artificial neural networks, in: Proceedings of ECAL'07, Springer-Verlag, 2007, pp. 936–945.
[41] P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Transactions on Neural Networks 5 (1993) 54–65.
[42] C.G. Miguel, C.F.d. Silva, M.L. Netto, Structural and parametric evolution of continuous-time recurrent neural networks, in: SBRN '08: Proceedings of 10th Brazilian Symposium on Neural Networks, IEEE Computer Society, 2008, pp. 177–182.
[43] D.B. D'Ambrosio, K.O. Stanley, A novel generative encoding for exploiting neural network sensor and output geometry, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation Series GECCO '07, ACM, New York, NY, USA, 2007, pp. 974–981. [Online]. Available: ⟨http://doi.acm.org/10.1145/1276958.1277155⟩.
[44] K.O. Stanley, D.B. D'Ambrosio, J. Gauci, A hypercube-based encoding for evolving large-scale neural networks, Artif. Life 15 (April (2)) (2009) 185–212. [Online]. Available: ⟨http://dx.doi.org/10.1162/artl.2009.15.2.15202⟩.
[45] Z. Buk, J. Koutník, M. Šnorek, Neat in hyperneat substituted with genetic programming, in: Proceedings of the 9th International Conference on Adaptive and Natural Computing Algorithms, ICANNGA'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 243–252. [Online]. Available: ⟨http://dl.acm.org/citation.cfm?id=1813739.1813766⟩.
[46] S. Risi, J. Lehman, K.O. Stanley, Evolving the placement and density of neurons in the hyperneat substrate, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO'10, ACM, New York, NY, USA, 2010, pp. 563–570. [Online]. Available: ⟨http://doi.acm.org/10.1145/1830483.1830589⟩.
[47] R. Poli, Evolution of graph-like programs with parallel distributed genetic programming, in: Proceedings of ICGA'97, Morgan-Kaufmann, 1997, pp. 346–353.
[48] J. Pujol, R. Poli, Evolving the topology and the weights of neural networks using a dual representation, Appl. Intell. J. 8 (1998) 73–84.
[49] R. Poli, M. Graf, Free lunches for neural network search, in: Proceedings of GECCO'09, ACM, 2009, pp. 1291–1298.
[50] I. Rechenberg, Evolutionsstrategie—Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution, Ph.D. Dissertation, Technical University of Berlin, Germany, 1971.
[51] V.K. Vassilev, J.F. Miller, The advantages of landscape neutrality in digital circuit evolution, in: Proceedings of ICES, Lecture Notes in Computer Sciences, vol. 1801, Springer Verlag, 2000, pp. 252–263.
[52] T. Yu, J.F. Miller, Neutrality and the evolvability of boolean function landscape, in: Proceedings of EuroGP, Lecture Notes in Computer Sciences, vol. 2038, Springer-Verlag, 2001, pp. 204–217.
[53] J.F. Miller, S.L. Smith, Redundancy and computational efficiency in cartesian genetic programming, IEEE Transactions on Evolutionary Computation 10 (2) (2006) 167–174.
[54] M.I. Jordan, Attractor dynamics and parallelism in a connectionist sequential machine, in: J. Diederich (Ed.), Artificial Neural Networks, IEEE Press, Piscataway, NJ, USA, 1990, pp. 112–127. [Online]. Available: ⟨http://dl.acm.org/citation.cfm?id=104134.104148⟩.
[55] A.P. Wieland, Evolving controls for unstable systems, in: Proceedings of Connectionist Models Summer School, 1991, pp. 91–102.
[56] J. Gomez, F. Schmidhuber, R. Miikkulainen, Efficient non-linear control through neuroevolution, in: ECML 2006: Proceedings of the 17th European Conference on Machine Learning, Springer, 2006.
[57] W.H. Wolberg, O.L. Mangasarian, Multisurface method of pattern separation for medical diagnosis applied to breast cytology, Proceedings of the National Academy of Sciences of the United State of America 87 (1990) 9193–9196.
[58] M. Iranpour, S. Almassi, M. Analoui, Breast cancer detection from FNA using SVM and RBF classifier, in: 1st Joint Congress on Fuzzy and Intelligent Systems, 2007.
[59] C.S.S.A. Túlio, R.M. Rangayyan, Classification of breast masses in mammograms using neural networks with shape, edge sharpness and texture features, J. Electron. Imaging 15 (1) (2006) 013019.
[60] R. Santo, R. Lopes, R. Rangayyan, Classification of mammographic masses using radial basis functions and simulated annealing with shape, acutance, and texture features, in: Proceedings of the IASTED Conference on Biomedical Engineering, 2005, pp. 458-175.
[61] H.D. Cheng, M. Cui, Mass lesion detection with a fuzzy neural network, Pattern Recognition 37 (6) (2004) 1189–1200.
[62] G. Palmer, C. Zhu, T. Breslin, F. Xu, K. Gilchrist, N. Ramanujam, Comparison of multi excitation fluorescence and diffuse reflectance spectroscopy for the diagnosis of breast cancer, IEEE Trans. Biomed. Eng. 50 (11) (2003) 1233–1242.
[63] M.K. Markey, J.Y. Lo, G.D. Tourassi, C.E. Floyd, Self-organizing map for cluster analysis of a breast cancer database, Artif. Intell. Med. 27 (2003) 113–127.
[64] S. Ho, H. Chen, S. Ho, T. Chen, Design of accurate classifiers with a compact fuzzy-rule base using an evolutionary scatter partition of feature space, IEEE Trans. Syst. Man Cybern. 34 (2) (2004) 1031–1044.
[65] L. Wei, Y. Yang, R.M. Nishikawa, Y. Jiang, A study on several machine-learning methods for classification of malignant and benign clustered microcalcifications, IEEE Trans. Med. Imaging 24 (2005) 371–380.
[66] M.M. Khan, G.M. Khan, J.F. Miller, Evolution of neural networks using cartesian genetic programming, in: IEEE Congress on Evolutionary Computation, 2010, pp. 1–8.
[67] H. Guo, A.K. Nandi, Breast cancer diagnosis using genetic programming generated feature, Pattern Recognition 39 (5) (2006) 980–987.
[68] H. Jianchao, N. Cercone, *uleViz*: a model for visualizing knowledge discovery process, in: KDD, 2000, pp. 244–253.
[69] J.C. Werner, T.C. Fogarty, Genetic programming applied to severe diseases diagnosis, in: Proceedings of Intelligent Data Analysis in Medicine and Pharmacology (IDAMAP-2001), 2001.
[70] R. Nandi, A. Nandi, R. Rangayyan, D. Scutt, Classification of breast masses in mammograms using genetic programming and feature selection, Med. Biol. Eng. Comput. 44 (8) (2006) 683–694.
[71] W.H. Land Jr., L. Albertelli, Y. Titkov, P. Kaltsatis, G. Seburyano, Evolution of neural networks for the detection of breast cancer, in: Proceedings of IEEE International Joint Symposia on Intelligence and Systems, INTSYS '98, 1998.
[72] T. Mu, A.K. Nandi, Breast cancer detection from FNA using SVM with different parameter tuning systems and SOM-RBF classifier, J. Franklin Inst. (2013) 285–311, http://dx.doi.org/10.1016/j.jfranklin.2006.09.005 in press.
[73] D. West, V. West, Model selection for a medical diagnostic decision support system: a breast cancer detection case, Artif. Intell. Med. 20 (3) (2000) 183–204.
[74] H.A. Abbass, An evolutionary artificial neural networks approach for breast cancer diagnosis, Artif. Intell. Med. 25 (2002) 265–281.
[75] S.-Y. Ho, H.-M. Chen, S.-J. Ho, T.-K. Chen, Design of accurate classifiers with a compact fuzzy-rule base using an evolutionary scatter partition of feature space, IEEE Trans. Syst. Man Cybern. B 34 (2) (2004) 1031–1044.
[76] W. Lam, C.-K. Keung, C.X. Ling, Learning good prototypes for classification using filtering and abstraction of instances, Pattern Recognition 35 (7) (2002) 1491–1506.
[77] R. Janghel, A. Shukla, R. Tiwari, R. Kala, Intelligent decision support system for breast cancer, in: Advances in Swarm Intelligence, Springer, Berlin, Heidelberg, 2010, vol. 6146, pp. 351–358 (Chapter 46).
[78] B.A. Garro, H. Sossa, R.A. Vzquez, Artificial neural network synthesis by means of artificial bee colony (abc) algorithm, in: IEEE Congress on Evolutionary Computation, IEEE, 2011, pp. 331–338.
[79] B.A. Garro, H. Sossa, R.A. Vazquez, Design of artificial neural networks using a modified particle swarm optimization algorithm, in: Proceedings of the 2009 International Joint Conference on Neural Networks, IJCNN'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 2363–2370.

[80] H.W. William, W. Street, O. Mangasarian, Image analysis and machine learning applied to breast cancer diagnosis and prognosis, Anal. Quant. Cytol. Histol. 17 (2) (1995) 77–87.
[81] H.H. Dam, H.A. Abbass, C. Lokan, X. Yao, Neural-based learning classifier systems, IEEE Trans. Knowl. Data Eng. 20 (January (1)) (2008) 26–39. [Online]. Available: ⟨http://dx.doi.org/10.1109/TKDE.2007.190671⟩.
[82] M.M. Islam, X. Yao, S.M.S. Nirjon, M.A. Islam, K. Murase, Bagging and Boosting Negatively Correlated Neural Networks, 2008.

Maryam Mahsal Khan. did her BSc Computer System Engineering from UET Peshawar, Pakistan and a Master from Universti Teknologi Petronas, Malaysia. She is currently a PhD student at the University of Engineering and Technology Peshawar, Pakistan. She has a keen interest in Non-Linear Control, Genetic Algorithms and Genetic Programming, Artificial Neural Networks, Pattern Recognition, Image Processing and Evolvable hardware. She has a range of publications in these fields in the conferences of repute.

Arbab Masood Ahmad. passed his BSc Electrical Engineering in 1990, from the University of Engineering and Technology, Peshawar, Pakistan. He joined Medical Engineering Department of Siemens Pakistan Engineering Company, the same year. He did his MSc Electrical Engineering from the University of Engineering and Technology Taxila, Pakistan in 1997, as part time student. After serving Siemens Pakistan for fourteen years, initially as a Technical service Engineer and later on as Deputy Manager Service, he left the industry and joined the academia. He served COMSATS Institute of Information Technology, Wah, Pakistan, from 2004-2008. He is currently an Assistant Professor in the Department of Computer Systems Engineering, University of Engineering and Technology Peshawar and also doing his PhD from the same university. The title of his research is “Bio-signal Processing using computational intelligence”. He has one journal and three International Conference publications.

Gul Muhammad Khan. did his BSc Electrical Engineering from UET Peshawar Pakistan with distinction and a PhD in the field of Intelligent System design in a record time of two years and eight months, from the University of York, UK in 2008. Since then he has been working as an Assistant Professor at the University of Engineering and Technology. Creativity and fast execution flow through his veins unquenched. His sharp eye toward innovation and the love he has for creative ideas comes from years of experience as a lead in various research projects and trainings he has had. Being a professional in the field of planning, he has also been a consultant for various electro medical equipment companies. He has a keen interest in Intelligent Smart Grid System Design, Genetic Algorithms and Genetic Programming, Wireless sensor networks and Evolvable hardware. He has a range of publications in these fields both in journals and conferences of repute. He has established a research centre "Center of Intelligent Systems and Network Research" at UET Peshawar, in march 2011, and is the Director of the same since then. Dr. Gul Muhammad Khan introduced two well known algorithms "CGP Developmental Networks" in the field of computational development and "CGP Neural Networks" in the field of Neuroevolution having numerous applications.

Julian. F. Miller. has a BSc in Physics (Lond), a PhD in Nonlinear Mathematics (City) and a PGCLTHE (Bham) in Teaching. He is an academic in the Department of Electronics at the University of York. He has chaired or co-chaired fifteen international workshops, conferences and conference tracks in Genetic Programming (GP), Evolvable Hardware. He is a former associate editor of IEEE Transactions on Evolutionary Computation and an associate editor of the Journal of Genetic Programming and Evolvable Machines and Natural Computing. He is on the editorial board of the journals: Evolutionary Computation, International Journal of Unconventional Computing and Journal of Natural Computing Research. He has publications in genetic programming, evolutionary computation, quantum computing, artificial life, evolvable hardware, computational development, and nonlinear mathematics. He is a highly cited author with over 4,000 citations and over 210 publications in related areas. He has given nine tutorials on genetic programming and evolvable hardware at leading conferences in evolutionary computation. He received the prestigious EvoStar award in 2011 for outstanding contribution to the field of evolutionary computation. He is the inventor of a highly cited method of genetic programming known as Cartesian Genetic Programming and edited the first book on the subject in 2011.