Graph Genetic Programming with Heuristics Based Crossover

Bogdan Burlacu*, Lavinia Ferariu*

**Department of Automatic Control and Industrial Informatics,*
"Gheorghe Asachi" Technical University of Iasi, Romania,
RO, 27 Bd. D. Mangeron 27, IS 700 050 (e-mail: bburlacu@ ac.tuiasi.ro, lferaru@ac.tuiasi.ro).

Abstract: This paper presents a new neuro-evolutionary system for the design of feed forward hybrid neural models. Every individual is directly encoded as a directed acyclic graph, allowing the development of partially interconnected architectures with heterogeneous layers including both local and global neurons. The use of different neuron types provides improved performances in terms of approximation quality for a large class of nonlinear system identification problems, while the direct graph encoding ensures a simpler interpretation of the hierarchical individuals, as well as a compact representation. The authors recommend a special minimally sufficient set of functions and customized compatible genetic operators, which exploit the inherent characteristics of the neural architectures (modularity, high interconnectivity and high level of parallelism). An extra effort is made towards improving the genetic algorithms convergence speed by introducing a special sensitivity score for each hidden neuron as part of a novel crossover strategy aimed to preserve valuable structural blocks inside the individual. Featuring multithreaded execution and specialized data structures, the C application scales well on multi core computers and has a low memory footprint, being suitable for complex nonlinear identification problems.

Keywords: genetic programming, genetic operators, neural networks, system identification, optimization.

1. INTRODUCTION

The employment of genetic programming as a neuro - evolutionary technique represents a modern approach for complex nonlinear identification problems. Neuro-genetic systems benefit greatly from the learning ability and computational efficiency (due to the highly interconnected, parallel structure) of neural networks, while, on the other hand, the evolutionary component provides a more robust and flexible selection of the model structure and/or corresponding parameters, being also able to deal with noisy data, time-variance, nonlinearities and other difficulties specific to industrial environments. Overall, one such system offers increased flexibility for the simultaneous selection of model structure and parameters for a wide range of applications, without requiring a large set of aprioric information (Flemming and Purshouse, 2002; Ferariu and Voicu, 2005; Poli et al., 2008).

With these advantages in mind, numerous approaches have been suggested in the related literature (Flemming and Purshouse, 2002). In each case, the first problem that arises is the selection of an appropriate chromosomial encoding of the hierarchical neural structures. The standard genetic algorithm requires the use of supplementary "control" genes for the encryption of hierarchical topologies, which leads to longer chromosomes and to a more difficult exploration, while the canonical GP implicitly works on tree-based individuals. The most direct and compact representations is the graph based one, yet this is not a common encryption in the standard evolutionary loop. The extension of canonical GP to work on graph-based individuals requires a more complicated logic inside the genetic operators, as well as specific measures to ensure the correctness of the resulting individuals.

This approach employs graph GP for the concomitant selection of neural topologies and parameters of **h**ybrid **n**eural **n**etworks (HNN). The modularity and parallelism of neural networks is exploited through a special definition of primitive sets and corresponding specific crossover and mutation operators that ensure that any recursive combination of functional and terminal nodes leads to valid individuals, from both a phenotypic and genotypic standpoint.

This paper introduces a special guided structural crossover which analyzes the performances of the component sub-graphs before selecting the cutting points. This heuristics is useful for preventing the loss of the best adapted substructures, with benefits on the performances of the resulted offspring and, consequently, on the convergence speed of the algorithm.

Unlike previous graph evolutionary algorithms which deal with a limited variety of structural templates (especially with MLP), the modular generation of graph-encrypted individuals enables the development of partially interconnected hybrid networks, which are expected to deliver increased performances for wide classes of applications (Flemming and Purshouse, 2002; Poli et al., 2008). The hybrid neural networks accept combinations of hidden neurons with global and local responses, thus being suitable for both interpolation and extrapolation problems (Ferariu and Voicu, 2005).

The paper is organized as follows. Section 2 browses the main features of the suggested design algorithm, whilst Section 3 discusses details regarding the proposed C implementation, with emphasis on data structure design and algorithms adopted for population initialization and offspring generation. Section 4 comments the applicability of the approach to chaotic time series prediction via several experimental trials and last section is devoted to conclusions.

2. ALGORITHM OVERVIEW

The algorithm starts with a randomly generated initial population of graph-based individuals, each one encrypting a possible hybrid neural model. The quality of each solution is assessed in terms of output squared error computed over the whole training data set and, subsequently, the fitness values, stated as selection probabilities, are determined by means of linear ranking. At each generation, the best individuals are selected into the recombination pool, by using stochastic universal sampling. Afterwards, specific crossovers and mutations are applied to produce new models, with potentially better performances. Note that the genetic operators act at both structural and parametric level. The best offspring replace the less adapted individuals in the population which is passed to the next generation.

The suggested design algorithm is compliant with feed-forward, partially interconnected HNN with external delay blocks. A neuron may receive input stimuli from any subset of neurons of the previous layer, as well as from any subset of the neural inputs. To provide enhanced adaption capabilities in interpolation and extrapolation problems, the hidden layers can comprise of any combination of neural units with local or global response. Note that the neurons with global response are able to bring increased generalization capabilities and robustness, as they aggregate their numerous inputs by means of a dot product operator, yet, the local neurons provide high computational efficiency due to the use of Euclidian distance input operator, which permits the activation of each neuron within a small area around a certain input sample.

The algorithm makes use of graph GP, namely it employs a direct encoding of neural architectures, by means of directed acyclic graphs (DAG). As opposed to the tree-based encoding involved in classic GP (Poli et al., 2008; Affenzeler et al., 2009), the DAG allows the encryption of each neural connection as a supplementary graph link, which leads to an efficient reuse of the available structural blocks and a significant reduction of memory consumption in the context of highly interconnected architectures (Walker and Miller, 2008). The suggested encoding accepts only variable – type terminals, placed on the leaves of the graph.

When series – parallel, input - output identification schemes are adopted, the terminals' set, **T**, may contain lagged plant input and output measurements

$$\mathbf{T} = [\mathbf{u}(k),..,\mathbf{u}(k-n_u),\mathbf{y}(k-1),..,\mathbf{y}(k-n_y)]\,,$$

where $\mathbf{u} \in \Re^m$ and $\mathbf{y} \in \Re^n$ denote the inputs and the outputs of the system, k indicates the current sampling instant and n_u, n_y represent the maximum permitted input and output lags, respectively.

To exploit the modularity of the neural topologies inside the structure of the hierarchical DAG-based individuals, each neuron is encrypted by a single node of the graph. The set of functions, **O**, contains the functions describing the input-output mapping performed by the accepted neural unit. Therefore, a simple extension to any desired hybrid neural architecture is permitted, yet, more important, one guarantees that any recursive combination between the elements of **O** and **T** encode a valid neural network. The functions of **O** accept variable arity and embed the corresponding neural parameters. The proposed encryption provides simple interpretations of the neural models, as well as compact representations. However, the genetic operators have to be reconfigured to work also on the inner structure of the functional nodes.

This paper considers the combination of global perceptrons with or without functional links, respectively, and local Gaussian neurons with real or complex weights. The functions' set results $\mathbf{O} = \{f_{PS}, f_{PF}, f_{GR}, f_{GC}\}$. Here,

$$f_{PS}(\mathbf{z},\boldsymbol{\theta}_{PS}) = \tanh(\sum_{i=1}^{no\_i} w_i z_i + b) \qquad (1)$$

corresponds to the standard perceptron (PS) having the inputs $[z_i]_{i=\overline{1,no\_i}}$ and the neural parameters

$$\boldsymbol{\theta}_{PS} = [w_1,..,w_{no\_i},b]' \in \Re^{no\_i+1}$$

(where w_i and b denote the weight of the i^{th} input connection and the bias, respectively) (Haykin, 1999), and

$$\begin{aligned} f_{PF}(\mathbf{z},\boldsymbol{\theta}_{PF}) = \tanh(\sum_{i=1}^{no\_i} \; (\sum_{j=1}^{P} w_{ij}^{c} (\cos \pi j z_i) \\ + \sum_{j=1}^{P} w_{ij}^{s} (\sin \pi j z_i) + w_i z_i + b)) \end{aligned} \qquad (2)$$

is associated to a global neuron with functional links (PF), assuming the orthogonal trigonometric expansion of maximum order P and the extended set of parameters $\boldsymbol{\theta}_{PF} \in \Re^{1+no\_i\cdot(2P+1)}$ (Patra et al., 1999). Also,

$$y_{GR} = f_{GR}(\mathbf{z},\boldsymbol{\theta}_{GR}) = e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{no\_i}(c_i - z_i)^2} \qquad (3)$$

describes the behaviour of a Gaussian neuron (GR) with real parameters $\boldsymbol{\theta}_{GR} = [c_1,...,c_{no\_i},\sigma]' \in \Re^{no\_i+1}$, namely the centres $[c_i]_{i=1,..no\_i}$, and the spread σ (Haykin, 1999) and

$$y_{GC} = f_{GC}(\mathbf{z}, \boldsymbol{\theta}_{RC}) = = e^{-w^2\left[\left(\sum_{i=1}^{no\_i}\cos\alpha_i \cdot (z_i - c_i)\right)^2 + \left(\sum_{i=1}^{no\_i}\sin\alpha_i \cdot (z_i - c_i)\right)^2\right]}. \qquad (4)$$

corresponds to a local neuron with radial basis function and complex weights (GC), which associates for each input connection a center, c_i and a complex weight, $w\,e^{\sqrt{-1}\cdot\alpha_i}$, leading to the set of parameters $\boldsymbol{\theta}_{GC} = [\alpha_1,..,\alpha_{no\_i}, c_1,...,c_{no\_i}, w]' \in \Re^{2no\_i+1}$ (Igennik et al., 2001).

Note that although many homogenous neural networks (such as MLP, RBF) have been proved to be universal approximators of nonlinear, bounded, continuous functions, no construction theorem is available to guarantee a certain approximation performance. Therefore, in practical situations, when dealing with noisy, large training data sets and neural input of high dimension, the design procedure might fail by selecting an unsatisfactory model. If an adaptive architecture configuration is provided, HNNs can lead to better approximation capabilities than the homogeneous neural networks (Ferariu and Voicu, 2005). However, previous work is limited to the design of HNNs with a single hidden layer, by means of fixed length chromosomes. By employing GP techniques which inherently work on individuals of various sizes and shapes, the present approach provides increased flexibility in neural architecture configuration.

For the sake of simplicity, the models are forced to include a single output neuron. In the case of multi-inputs multi-outputs systems, one can design a distinct neural model for the approximation of each system output, due to the fact that **T** permits to illustrate any possible interdependencies between the plant variables. To ensure non-zero response for a large range of neural inputs, the output neuron is assumed to be global. If lagged values of state variables are included in **T**, the approach can be used for the design of state based models for multivariable nonlinear systems.

Note that **O** and **T** satisfy the closure property (Koza, 1992), as data type consistency and data values consistency are ensured for any possible graph. Consequently, any individual is valid from GP perspective. Moreover, because no other types of nodes are necessary for producing the optimal solution, **O** is minimally sufficient and, if n_u and n_y are chosen large enough, **T** results sufficient, too. Obviously, GP can cope with several alien terminals, due to its capacity of selecting a subset of **T** inside each generated individual, so it is not compulsory to use minimum n_u and n_y, therefore these parameters can be simply tuned by trial and error.

3. IMPLEMENTATION DETAILS

The application uses an explicit representation of nodes/neurons as a combination of nested data structures, with emphasis on a clear distinction between functions and terminals. Following a recursive pattern, Terminal and Function data structures are conveniently nested inside the more complex Node structure, allowing the construction of N-ary trees and a more efficient implementation of the genetic operators. The Function structure includes parameters which help keeping the track of nodes' children stored and managed by means of a dynamically allocated array of pointers to Node structures. Additionally, specific data structures allow operations over an individual from a neural network perspective, enabling it to be seen as a multilayered architecture. This is achieved by grouping together the tree nodes found at the same depth level and by allowing the traversing of such individuals layer by layer, instead of the conventional stack (queue) based or recursive techniques. The NodeArray structure keeps track of all tree nodes for a given depth, while the NeuralNet structure holds multiple layers (Node arrays), thus making possible to visit all inner nodes with two iterators, once a tree individual has been ”layerized”.

The stochastic procedure uses the glib random number generator which uses the Mersenne Twister PRNG, which was originally developed by Makoto Matsumoto and Takuji Nishimura. Further information can be found at http://www.math.sci.hiroshima-u.ac.jp/~m-at/MT/emt.html.

Suitable functions provide an interface with the random source, to allow the selection of Functions and Terminals, as well as the initialization of the other node parameters (such as bias, weights, lags, etc). Note that the use of the linux random number source devices (/dev/random and /dev/urandom) is also possible (due to an abstract interface), with the possibility to use the processors hardware number generator with the systems entropy pool.

When a new individual is created, its root node will have to be a global Function node. The depth value is randomly chosen and then, for each node, the type is stochastically generated, along with its corresponding parameters (e.g. number of children). Afterwards, the resulted tree-based individual is translated to a layered representation, via a procedure which joins together the pointer arrays inside the Function structures. The nodes from successive layers are randomly interconnected and the resulting connections are maintained for the life time of the individual.

A distinction is made between the normal, direct connections which are established when a Function node is initially created (towards nodes which are regarded as its children), and the extra-connections that are established afterwards, between the Function node and the children of other nodes situated at the same level. This distinction is important because it allows several optimizations (e.g., the recursive procedures can visit each node only once) and helps the crossover to swap the selected sub-graphs.

As the main concept of the application is the modular development of the neural networks by means of GP-based optimization, one has to guarantee that, during the evolutionary loop, each graph-based individual encodes a valid configuration of neurons and terminals. In this context, validity means that the main traits of a neural network must be preserved (e.g., formally approved/established activation functions, along with neural parameters such as bias/dispersion of neurons, weights of incoming connections, etc). The activation functions have to be implemented according to a predefined calling convention, by involving another customized data structure, NodeInputs, which is used to pass functions' parameters whenever the evaluation of an individual is performed. To support a modular design of the neural network and a simple management of different types of neurons, the program makes use of a dispatch table, namely a function pointer array which holds the addresses of all allowed activation functions. In this configuration, adding a new type of neuron is trivial, as it all comes down to defining the new activation function and adding it to the dispatch table. This extra flexibility comes with the cost of an indirect call of the activation functions, which might translate into small performance penalties, but most modern x86/x64 CPUs can perform branch prediction on indirect calls, so the effect should be minimal and barely noticeable, if any. On the other hand, a dispatch table offers the flexibility of adding or removing such functions, without having to change other areas of the program.

The GP application is organized into several sub-modules which perform operations on Terminals and Functions, Nodes (of one of the two types), DAG-encoded Individuals and populations of individuals. This hierarchical organization lends itself to good scalability, making the program able to process very large graphs (with numerous nodes/interconnections), as basis for delivering good results for a variety of optimization problems. Additionally, the genetic operators are required to work on highly interconnected neural structures. Therefore, the design procedure has to be able to process a large number of incoming and outgoing neuron connections, the goal being to reuse as much information as possible in the resulted DAG-based offspring.

In the case of structural crossover, randomly selected sub-graphs are swapped between the parents. This requires specific preparations for the sub-graph externalization, which basically means that its connections with the rest of the parent's nodes must be replaced in both directions (incoming and outgoing). Let us denote the root of a sub-graph with *G*. The algorithm relies on the following statements: (S1) a node *N* is called directly included in *G* sub-graph, if there exists a path formed only with normal connections provided from *G* to *N*; (S2) a node *M* is named indirectly included in graph *G*, if it is not directly included in *G* and there is a path of normal and extra-connections from *G* to *M*. Fig. 1 explains the crossover algorithm, assuming that the sub-graph having the root node 7 has been selected for swapping with the second parent. Firstly, according to (S1) and (S2), the procedure identifies the subset of nodes indirectly included in the selected sub-graph (marked with in red Fig 1). Afterwards, following a one-to-one mapping, a hash table is created, having the subset nodes as keys, and copies of those nodes as values. From the sub-graph's perspective, in hash table terms, all outgoing connections towards keys must be replaced with connections towards values, so that the sub-graph will carry within itself the same information when swapped, without disrupting the configuration that is left behind. The replacement procedure employs special checks in order to preserve the consistency of the DAG-encoded individual, because from the sub-graph's perspective, in some cases extra connections towards the keys must become normal connections towards values and, at the same time, the values can be connected together themselves, with either a normal or an extra connection.

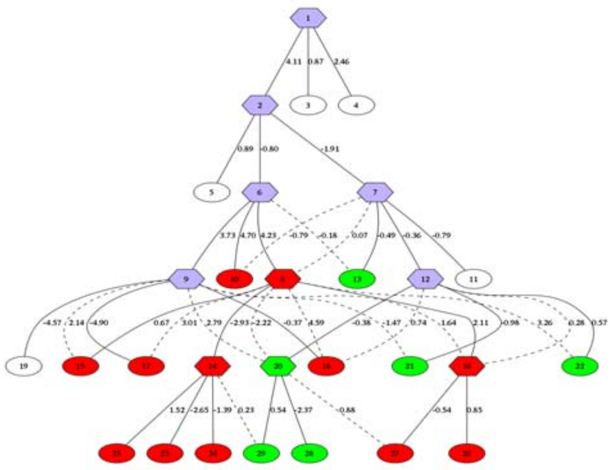

Fig. 1. DAG-encoded individual selected for crossover. The neurons are marked with hexagons, the terminal with ellipses; the normal connections are denoted with solid lines and extra-connections (introduced at "layerization") are represented by dotted links.

Let 8' be the copy of node 8 (the value associated to the key), node 10' be the copy of node 10, and so on. The extra-connections from node 7 towards nodes 8 and 10 will be replaced with normal connections towards nodes 8' and 10'. Then, node 8' will be connected with node 18'. However, since node 12 also has an extra-connection towards node 18', in order to preserve the consistency of the encoding, one of the nodes (node 12) will be connected to node 18' using a normal connection, while the other one (node 8') will have an extra connection towards node 18'. The procedure stops when all nodes in the sub-graph and all nodes in the hash table have been visited, and all connections have been replaced.

Once the algorithm has worked out the connections that need to be replaced from the sub-graph's standpoint, it switches to an "outside" view, identifying those nodes belonging to the selected sub-graph, which have incoming connections from nodes not belonging to the selected sub-graph. These connections will have to be replaced as well, in order to completely isolate the selected sub-graph and allow it to be swapped. The subset consisting of nodes 13, 20, 21, 22, 28, 29 is identified and again, a hash table is used. Applying the same replacement rules, node 6 will be connected with node 13' (the connection will no longer be an extra connection but a normal one), node 8 will be connected with node 20', and in turn, node 21' will have to be connected to nodes 28' and 29', using normal connections. Node 14 will receive an extra connection towards node 29' and nodes 9 and 12 will receive extra connections towards node 20'. Finally, node 9 is connected to nodes 21' and 22', using normal connections. Afterwards, the crossover involves an exchange of pointers from one side to the other.

It is already known that the crossover has the tendency of producing larger and larger individuals, even without ensuring significant improvement of the objective values. This phenomenon, called bloat, may lead to the production of overfitted individuals, with expected bad generalization capabilities. As a preliminary protection of individuals' parsimony, the application eliminates all the solutions which exceed a predefined depth. Note that, additionally, due to the way the extra-connections could be processed within the DAG- based individuals, the structural crossover might lead to a "horizontal expansion", displayed by increased number of normal links and, therefore, increased number of nodes. Another downside of the crossover is the competing conventions problem. It refers to the possibility of producing less adapted offspring, when working on different fitted parents, which encode the same neural network (for instance, when some neural building blocks are permutated in the selected DAGs).

In this context, the paper suggests an additional heuristic which can be applied prior to crossover in order to identify the neurons from the hidden layers of an individual which are most relevant to the individuals' performances. For this purpose a set of 'sensitivity scores' are computed, based on the following algorithm. For the individuals with the accuracy better than the average, sensitivity scores are assigned to their hidden neurons:

$$C^i = w_i \cdot \sum_{k=1}^{N} \left| \frac{{y_{hid}}^i(k)}{err_{out}(k)} \right|, \tag{5}$$

where $y^i_{hid}(k)$ represents the output of the i^{th} hidden neuron, w_i represents the connection weight from the i^{th} hidden neuron to its parent neuron on the upper layer and $err_{out}(k) = y_{out}(k) - y_{ref}(k)$ represents the output error corresponding to sample k. Each hidden neuron that has a sensitivity score better than average, along with its inputs (in effect, a whole subgraph) is considered valuable and 'locked' so that it won't be affected by crossover.

By encapsulating the better adapted substructures, it is possible to preserve the most relevant genetic material of the parents within the generated offspring, therefore enforcing the convergence speed of the algorithm. Targeted to reduce the horizontal expansion tendency and the bloat phenomenon, the proposed GP application also considers different types of mutation, applied with higher probabilities. Being unary operators, the mutations permit a better control on the complexity of the resulted offspring, and avoid competing conventions' problem occurrence.

The parametric mutation randomly selects a node and alters the values of its corresponding parameters, whilst the link mutation adds extra-connections from function nodes/neurons to the children of other neurons situated on the same level.

The node mutation changes the type of randomly selected nodes. When a Function node turns into a Terminal one, some extra consideration has to be given to the connections which might exist from outside nodes towards the children of the Function node, which will be structurally mutated. As the children of the node will no longer exist, all references from the outside nodes towards them must be removed, or copies must be provided.

Lastly, the structural mutation replaces a stochastically selected sub-graph with a randomly generated one.

The evaluation procedure performs a recursive traversal of a DAG-encoded individual, by propagating the results of the computations upward, from the bottom of the graph towards the root node. When a Terminal is reached, the corresponding value is read based on its lag, the input label and the current sampling time. After all terminals return their values, the evaluation procedure calls the activation function for the Function node that they are connected to and the result is stored in a variable. Since the procedure is dealing with a large number of extra connections, a simple cache system is used: it sets an appropriate flag when a node is firstly visited, so that ulterior visits will retrieve the already saved result without wasting CPU time by re-evaluation.

Because during a GP generation, one of the most important time consumers is the computation of the objective values, the application uses pthreads [https:// computing.llnl.gov/ tutorials/ pthreads/ #Overview] to distribute this task among the available CPU cores. Depending on the number of existing cores, the population (a NodeArray structure) is partitioned, by means of indexes and offset variables, so that each thread can perform the calculations independently (the threads are joined after evaluation of every individual).

Additionally, the program can export tree individuals to plain text files using the dot language to represent graph structure (http://www.graphviz.org). Using the opensource Graph Visualization Software (Graphviz), the files can be converted to a wide variety of graphic formats.

4. APPLICATION

The software package has been verified on Lorenz attractor time series prediction. The training and validation data sets, each one consisting of 250 samples, indicate different state trajectories collected for distinct initial conditions ([0.1,0.1,0.1] and [0.3,0.3,0.3], respectively), considering the chaotic state model $\dot{x} = 10(y - x)$, $\dot{y} = -xz + 27x - y$, $\dot{z} = xy - 8/3 * z$ and the sampling period $T_S = 0.1$. The HNN is designed to predict x trajectory. Therefore, $\mathbf{T}$ contains lagged values corresponding to all system state variables, $\mathbf{T} = [x(k-1), y(k-1), z(k-1), .. \; y(k-n_y), z(k-n_z)\,]$.

The main challenge for the application lies in delivering the high performances of accuracy and generalization required for the selected model, in compliance with the chaotic nature of Lorenz system. Several algorithm parameter sets were used to illustrate the behavior of the proposed GP-based approach (Table 1). The population was initialized with simple HNNs having maximum three incoming connections per neuron, as the interconnectivity may significantly increase during the evolutionary loop, by means of genetic operators. Table 1 indicates the average accuracy performances, as well the quality of the best model resulted during 10 independent runs. Here, *MSE_L* and *MSE_T* represent the mean output squared errors obtained over scaled learning and testing data, respectively. For the sake of simplicity, one considers $n_x = n_y = n_z = n$.

Table 1. Experimental Results – Crossover without Additional Heuristics

#	N_{ind}/ N_{gen}	n	d	Average values		Best model	
				MSE_L	*MSE_T*	*MSE_L*	*MSE_T*
1	1000/300	1	3	7.2	1.3	3.22	0.41
2	1000/300	3	3	1.1	1.69	0.31	0.54
3	5000/300	1	3	3.41	0.63	0.78	0.38
4	5000/300	3	3	0.76	1.04	0.39	0.44
5	10000/300	1	3	2.3	0.36	0.40	0.24
6	10000/300	1	4	0.94	0.27	0.63	0.34

Nind/Ngen = population size/ number of generations; *n*=maximum lag; *d*= maximum depth of graphs.

Table 2. Experimental Results – Crossover with Additional Heuristics

#	N_{ind}/ N_{gen}	n	d	Average values		Best model	
				MSE_L	*MSE_T*	*MSE_L*	*MSE_T*
1	1000/300	1	3	0.946	0.999	0.481	0.520
2	1000/300	3	3	1.164	1.247	0.441	0.422
3	5000/300	1	3	0.602	0.657	0.399	0.413
4	5000/300	3	3	0.570	0.643	0.213	0.231
5	10000/300	1	3	0.361	0.356	0.295	0.304
6	10000/300	1	4	0.349	0.339	0.332	0.308

Nind/Ngen = population size/ number of generations; *n*=maximum lag; *d*= maximum depth of graphs.

Firstly, the application was verified without including the suggested heuristics based crossover. If $n = 1$, $\mathbf{T}$ results minimally sufficient (#1, #3) and the risk of overfiting is reduced. Although it seems simpler to find proper models by exploring fewer potential neural topologies, note that the individuals' performances reveal the symbiosis between structure and parameters, so GP can discard HNNs with appropriate architecture, yet inconvenient parameters. Consequently, the final model could be sometimes simpler than necessary, leading to improper *MSE_L*. This downside could be diminished when working on larger populations (#1 vs. # 3, #5) and, obviously, the hybridization with certain local optimization procedures could be beneficial. If $\mathbf{T}$ includes alien lagged state variable values ($n > 1$), the use of small d permits to eliminate excessively complex individuals expected to include unnecessary intron building blocks, yet the risk producing overfitted individuals remains. However, the best individual came up during run #4, which proves that the genetic algorithm can also work on slightly larger terminal sets.

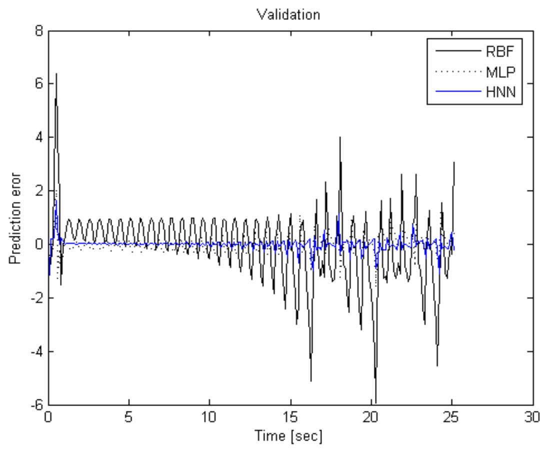

Fig. 2. The prediction error provided by HNN /MLP/ RBF on non-scaled validation data set. Resulted mean output squared errors are, correspondingly, 15.08 / 35.2 / 458.6.

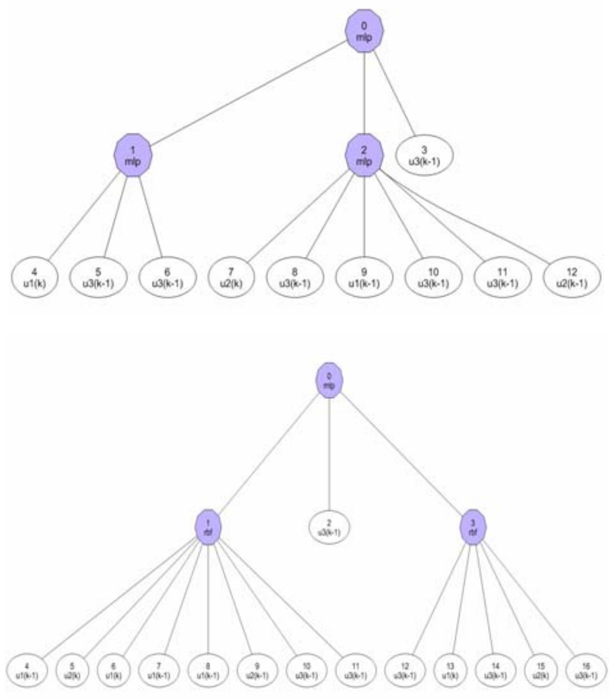

Fig. 3. Individuals generated according to configuration #3 indicated in Table 2.

The exploration can be also enforced by working on larger and diverse populations, although increasing the population size (and/or the number of generations) is not always a guarantee for performance improvement.

Stress testing (#5, #6) reveals the capacity of the algorithm of working on large batches of neurons (about 230000 per generation), whilst preserving reasonable time performances (about 120-200 sec total execution time) and memory consumption (about 65 MB). All trials were carried out on a configuration with Code 2 Duo P7350 (2GHz, 3MB cache) and 3GB RAM. Note that each Function data structure needs 40 bytes and each Terminal structure requires 12 bytes.

Lastly, the performances of the selected model #4 (including one PS output neuron and 2 hidden neurons – 1 PS and 1 GR) were compared with those provided by homogeneous neural networks having the same number of hidden neural units: an MLP trained for 5000 epochs with Levenberg-Marquardt algorithm, and an RBF designed by means of a constructive algorithm (which iteratively adds GR neurons). The designed HNN features better prediction and better generalization capabilities (Fig.2), making use of its particular compact partially interconnected heterogeneous structure. Comparable performances of accuracy and generalization can be achieved with larger homogeneous neural networks, including about 4 hidden PS, or 15 GR.

When using the heuristics based crossover (Table 2) certain building blocks belonging to the best individuals were preserved by not allowing the genetic operators to act on them if their sensitivity score was greater than a certain ratio. This enhanced crossover has improved the convergence of the GP system by allowing good structures/sub-graphs to remain unchanged.

The results listed in Table 2 indicate better accuracy on the training data set for the neural model selected in most of the configurations (#1, 3, 4, 5, 6 in Table 1 vs. #1, 3, 4, 5, 6 in Table 2 for average *MSE_L* and *MSE_L* of the best designed model). However, due the fact the encapsulation of the best adapted substructures does not take into account the complexity order, *MSE_T* is not necessarily smaller (#3 in Table 1 vs. #3 in Table 2). Although note that even in these cases the models admit a reduced number of parameters, as illustrated in Fig. 3. The main explanations refer to the flexibility allowed by the partial interconnected HNN formalism and the suggested enhanced GP techniques.

5. CONCLUSIONS

The suggested C-based approach implements GP techniques in an efficient manner for the flexible design of hybrid neural network models. It achieves its goal by concomitantly working on the structure and parameters of the model, with the help of special genetic operators designed to ensure the validity of all models from both a phenotypic and genotypic standpoint. Addressing the validity issue involves keeping track of all node parameters, normal and extra connections inside the graph individual, prior and after the execution of mutation and crossover.

Furthermore, by using a simple heuristic based on node sensitivity scores to guide the crossover process, the overall convergence speed of the algorithm is improved.

The application features a low memory footprint and a fast evaluation of individuals, even when working on massive and highly interconnected structure. The experimental trials indicate the ability of the suggested approach to solve difficult identification/ modelling problems, while dealing with scarce a priori information and severe requirements of accuracy.

REFERENCES

Flemming, P.J. and Purshouse, R.C. (2002). Evolutionary Algorithms in Control Systems Engineering: A Survey, *Control Engineering Practice*, 10, 1223-1241.

Ferariu, L. and Voicu, M. (2005). Nonlinear System Identification Based on Evolutionary Dynamic Neural Networks with Hybrid Structure, *Proc. of IFAC Congress*, Prague, Czech Republic.

Poli, R., Langdon, W. B. and Mc Phee, N. F. (2008). *A Field Guide to Genetic Programming*, http://lulu.com (with contributions of J.R. Koza), [Online]. Available: http://www.gp-field-guide.org.uk.

Affenzeler M., Winkler S., Wagner S. and Beham A. (2009). *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications (Numerical Insights)*, CRC Press.

Walker A. and Miller, J.F. (2008). The Automatic Acquisition, Evolution and Reuse of Modules in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 12 (4), 397-417.

Haykin, S. (1999). *Neural Networks - A Comprehensive Foundation*, McMillan College Publishing Company, New York, 2nd Edition.

Patra J., Pal R., Chatterji and B., Panda G. (1999). Identification of nonlinear dynamic systems using functional link artificial neural networks, *IEEE Transactions on System, Man and Cybernetic*s, Part B: Cybernetics, 29, 254–262.

Igennik B., Tabib–Azar M., Le Clair S.R. (2001). A net with complex weights, *IEEE Transactions on Neural Networks*, 12, 236–249.

Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selectio*n. Cambridge, MA: MIT Press.