An improved method for automatically searching near-optimal artificial neural networks

Leandro M. Almeida, Teresa Ludermir

Abstract— **This paper describes an improved version of a method that automatically searches near-optimal Multilayer feedforward Artificial Neural Networks using Genetic Algorithms. This method employs an evolutionary search for simultaneous choices of initial weights, transfer functions, architectures and learning rules. Experimental results have shown that the developed method can produce compact, efficient networks with a satisfactory generalization power and with shorter training times when compared to other methods found in the literature.**

I. Introduction

The search for an Artificial Neural Network (ANN) tailored to a specific problem is an extremely important task for attaining success in an application that uses ANNs. This search is generally performed by an ANN configuration developer through a trial-and-error procedure. Thus, optimality or even near-optimality is not guaranteed, as the space explored is only a small portion of the entire search space and the type of search is rather random [1]. An optimal neural network can be seen as an ANN tailored to a specific problem, thus having a smaller architecture with faster convergence and a better generalization performance [2], [3], [4]. The specific and correct (near-optimal) configuration of an ANN model for a certain problem through trial-and-error is considered a tedious, less productive and error-prone task [2], [3]. When the complexity of the problem domain increases and when near-optimal networks are desired, manual searching becomes more difficult and unmanageable [2]. The construction of near-optimal ANN configurations involves difficulties such as the exponential number of parameters that need to be adjusted; the need for *a priori* knowledge of the problem domain and ANN functioning in order to define these parameters; and the presence of an expert when such knowledge is lacking [3].

An automatic method can be used to avoid the problems stemming from a manual search. A significant amount of the methods found in the literature for searching near-optimal ANNs employ evolutionary techniques, specifically Genetic Algorithms (GA), whereas others employ non-evolutionary techniques. Some methods use evolutionary techniques to search ANN models with architecture optimization, as presented in [5], [6]. Another methods perform searches including more information, such as transfer functions, initial weights and learning rules (or learning algorithms), as presented in [2], [1]. There are also methods that employ non-evolutionary techniques that prune connections considered less significant [7], [8] or freeze weights when the same inputs are submitted to the network [7]. Methods using non-evolutionary techniques are focused on the manipulation of ANN architectures. Thus, they only investigate restricted topological subsets, whereas methods using evolutionary techniques search the complete class of network architecture, thereby achieving better results [1].

Leandro M. Almeida and Teresa Ludermir are with the Center of Informatics, Federal University of Pernambuco, P.O. Box 7851, Cidade Universitária, Recife - PE, Brazil, 50732-970, (email: {lma3,tbl}@cin.ufpe.br).

In this work, we present an improved version of a method developed for searching near-optimal networks using ANNs and GAs with direct encoding, denominated NNGA-DCOD (aNN + GA - Direct enCODe), as presented in [3]. The main improvements to NNGA-DCOD are related to the change in the selection criterion and the assessment of solutions/ANNs found. In previous versions, NNGA-DCOD with roulette wheel selection failed to achieve adequate results due to the selection criterion being too severe and discarding complex solutions/ANNs, even when they exhibited satisfactory performance. Other selection criterion was adopted and a revision/adjust in the assessment of solution and genetic operators was performed in order to improve the method in search for more complex ANNs configurations, i.e. with more hidden layers. Another significant improvement is the time consumption analysis of NNGA-DCOD. Although rarely described in papers on the search for near-optimal ANNs, this analysis is useful for determining the amount of time the method needs to perform the search. NNGA-DCOD differs from other methods in the search for ANNs with high performance, low complexity and training up to five epochs, and its evolutionary search uses individuals with direct encoding (e.g., real values) [3]. Therefore, the main contribution of this work regards the genetic operators specially developed to work directly with ANN information on individuals. Another contribution is the search for near-optimal neural networks that considers performance, simplicity and processing speed. This paper is organized as follows: Section 2 presents the NNGA-DCOD method; Section 3 describes experimental results, including time consumption analysis; Section 4 summarizes our conclusions and presents future work.

II. NNGA-DCOD Method

The development of the NNGA-DCOD was performed using Evolutionary ANNs (EANNs), a framework that makes possible the search for all ANN components needed for its functioning, as defined by Xin Yao [4]. Such a framework is composed of a combination of GAs and ANNs, allowing the exploration and exploitation of a large number of necessary aspects or components necessary to the building

978-1-4244-1821-3/08/$25.00©2008 IEEE

of well-performing ANN models, sucha as initial weights, transfer functions, topology setups and learning rules (training algorithm parameters) [4]. EANNs include a sequential layer search process, in which each layer has specific ANN information to be found by a specific GA. This information is organized as follows [4]:

- the evolutionary search for initial weights occurs in the lower layer;
- the evolutionary search for learning algorithm parameters is performed in the intermediary layer;
- the evolutionary search for architecture configurations is performed in the higher layer, including the number of connections, number of hidden nodes per layer, number of hidden layers, transfer functions per layer or hidden nodes.

Figure 1 displays a possible search configuration in which the evolution speed is faster with the layer of initial weights than with the other layers. This is a consequence of the high dimensionality of the exploration space for initial weights due to the lack of a priori knowledge on excellent sets of initial weights. The higher search layers search for architectures and learning algorithm parameters, as they have more a priori knowledge and allow the restriction of a more specific search space. There is a GA in each layer that works with a population of individuals related to the information from the current layer. A recent work used EANNs with the layer configurations in Figure 1 for searching near-optimal ANNs [2]. Thus, a more intensive search for architectures and initial weighs is performed based on the specified learning algorithm parameters.

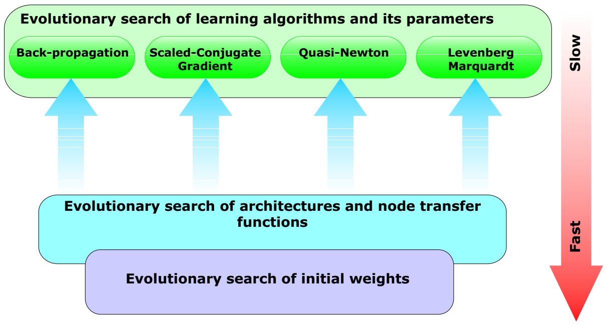
Evolutionary search of learning algorithms and its parameters
Back-propagation
Scaled-Conjugate Gradient
Quasi-Newton
Levenberg Marquardt
Evolutionary search of architectures and node transfer functions
Evolutionary search of initial weights
Slow
Fast

Fig. 1. Evolutionary search in layers.

Unlike the Abraham implementation [2] in which ANN information is encoded in binary form, the developed method uses real values, thereby avoiding the frequent encoding and decoding tasks at each iteration of the search process. The NNGA-DCOD performs the search for MLP, feedforward, fully connected ANNs, with supervised learning for classification problems that have a simple architecture (few hidden layers and nodes per layer), faster convergence into fewer training epochs and with satisfactory generalization performance. For example, in a work by Abraham [2], up to 500 training epochs are used for each network, whereas the NNGA-DCOD uses up to five training epochs. As the NNGA-DCOD uses real values rather than binary encoding schema, the traditional genetic operators are reformulated to deal with such values (ANN information).

The evolutionary layered search process displayed in Figure 1 is used in the NNGA-DCOD. The layer arrangement is the same used by [2], where the search for initial weights is performed in the lower layer; the search for hidden layers, nodes per layer (architecture) and transfer functions occurs in the intermediated layer; and the search for learning algorithm parameters is performed in the higher layer. Thus, there is a separate GA for each layer and, consequently, one population of individuals for learning algorithm parameters (PLAP), one for architectures and transfer functions (PATF) and another for initial weights (PIW).

As this is a sequential search process, three nested data structures are used for individuals in the NNGA-DCOD: the first is composed of a set of parameters for the training algorithms and a population from the second data structure. The second is composed of a set of architectures and a population from the third data structure. Finally, the third data structure is composed of a set of initial weights as presented (Figure 2).

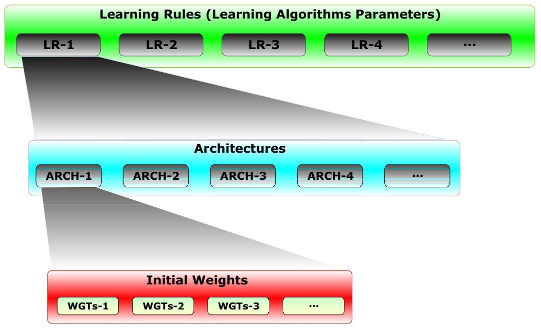

Fig. 2. Composition of data structure used.

As stated above, the NNGA-DCOD uses individuals with real values. Therefore, individuals from PIW are specific weight matrices, the encoding and decoding of which could present problems related to the representation of number precision for large float numbers. On the other hand, the adoption of real values to avoid this problem gives rise to the need for genetic operators that deal directly with real ANN information. The construction of genetic operators for individuals from PLAP, PATF and PIW was performed in stages with several experimentations in order to obtain the current operator versions. These operators are the main contribution of this work, as there is an absence of such types of genetic operators in the literature.

The selection of n individuals is performed using the tournament strategy, with a random pressure rate of $p = 50\%$ and an elitism rate of $e = 10\%$. This information was found empirically. The tournament selection operator is used to select individuals to compose a new population (survivors) as well as individuals for crossover.

One PIW is associated to a fixed pre-defined architecture in which all individuals have weight matrices with the same dimensions. Thus, the crossover of previously selected $\mathbf{piw}_n$ individuals, in which every weight matrix w has $c \times r$ dimensions, occurs as described in Algorithm 1. The idea of slicing the two weight matrices to form the child is related to maintaining the correlations that exist between weights. One portion of correlations between weights from two matrices will compose the child weight matrix. Therefore, from two matrices $\mathbf{a}$ and $\mathbf{b}$, half of $\mathbf{a}_l$, $\mathbf{a}_r$, $\mathbf{b}_l$, $\mathbf{b}_r$ is obtained, from which the new matrix will be defined, for example, $\mathbf{w}_{child} = [\mathbf{a}_l; \mathbf{b}_r]$ or $\mathbf{w}_{child} = [\mathbf{a}_r; \mathbf{b}_l]$.

Algorithm 1: Crossover Operator for PIW individuals

```
   Data: $\mathbf{piw}_n$, $w$ //parents, number of matrices in the network
   Result: childVet //new offspring
1  begin
2      $numChilds \leftarrow 1$
3      while $numChilds \leq (n/2)$ do
4          for $matrix \leftarrow 1 : w$ do
5              halfA $\leftarrow doSlice(\mathbf{pw}(numChilds).(matrix))$
6              halfB $\leftarrow doSlice(\mathbf{pw}(numChilds + 1).(matrix))$
7              $child(matrix) \leftarrow$ [halfA; halfB]
8          childVet $\leftarrow child$
9          $numChilds \leftarrow numChilds + 2$
10 end
```

The selection of individuals for mutation is performed randomly, as individuals have no fitness yet. According to the mutation rate $m = 40\%$, forty percent of the child will undergo mutation and this mutation will affect forty percent of its composition as well. Sparse matrices are generated for individuals selected for mutation, with forty percent of the values between $fx = [-0.5, 0.5]$ and the remaining values zeros. This range of values was also found empirically. These sparse matrices are then added to the child weight matrices. The fitness for PIW individuals is the normalized mean squared error (NMSE) from the training data submitted to the network.

For individuals from PATF, the fitness I_{fit} is composed of four pieces of information, I_{acc} classification accuracy, I_{nmse} training error, I_{comp} network complexity and I_f weight of transfer functions used.

$$I_{fit} = \alpha * I_{acc} + \beta * I_{nmse} + \gamma * I_{comp} + \delta * I_f \quad (1)$$

$$I_{acc} = 100 * \left(1 - \frac{correct}{total}\right) \quad (2)$$

$$I_{nmse} = \frac{100}{N\,P} * \sum_{j=1}^{P} \sum_{i=1}^{N} (d_i - o_i)^2 \quad (3)$$

$$I_{comp} = \frac{c}{c_{total}} \quad (4)$$

$$I_f = \sum_{h=1}^{n} fa_h \quad (5)$$

In the Equation 1, I_{acc} is the classification error percentage; I_{nmse} is the training NMSE generated by the network; I_{comp} is the complexity measure considering the number of used connections c and and the total number of possible connections c_{total}; I_f computes the weight from the transfer functions used. For such, every transfer function has a associated weight empirically found (Table I): P with 0.2, T with 0.4 and L with 0.7, prioritizing simple transfer functions, as the aim of the NNGA-DCOD is to find simple networks with high performance; N and P are the total number of outputs and number of training patterns, respectively; d and o are the desired output (target) and the network output (obtained), respectively. As the NNGA-DCOD searches for networks with up to three hidden layers, the c_{total} value is based on the amount of input (in), output (out) nodes and the number of hidden nodes per layer (hid): $c_{total} = in \times hid + hid + hid \times out$. In this work, the *winner-takes-all* classification criterion was adopted, in which the output node that has the highest value will determine the class pattern. For this, the number of output nodes is equal to the amount of the problem class. The I_{acc} calculus (network classification error) occurs through the *winner-takes-all* criterion.

Constants α, β, γ and δ have values between $[0, 1]$ and control the influence of the respective factors upon the overall fitness calculation process. For example, to favor the classification accuracy regarding training error and complexity, the constants are defined as follows: $\alpha = 1$, $\beta = 0.90$, $\gamma = 0.90$ and $\delta = 0.20$. These definitions imply that, when apparently similar individuals are found, those that have the least training error, structural complexity and transfer function complexity will prevail. These values were found empirically and used in the NNGA-DCOD experimentation.

The search for architectures is performed to find configurations with between one and three hidden layers and between one and 12 hidden nodes per layer. Table I displays the transfer functions enclosed in the search process. The selected individuals ("parents") are submitted to the crossover process performed according to Algorithm 2. The process starts with the definition of the number of hidden layers that the child will have, which is obtained through the mean sum of hidden layers from the parents, rounded off based on probability problems. The dimensions of the hidden layers and the transfer functions for the child are then defined through random selection that considers all parent information, hidden layers and transfer functions.

The crossover operator for PATF produces individuals that are very similar to their parents. Thus, the mutation operator (Algorithm 3) is applied after the crossover in order to maintain diversity in the population. Child selection for mutation is performed through a random process. The process starts with the number of children that will undergo mutation. A random number is then generated within the range $[0, 1]$. This value will be used to define whether an architecture will undergo an increase or decrease in the number of hidden layers or will undergo an increase ($pInc = 0.6$) or decrease ($pDec = 0.5$) in the number of hidden nodes per layer. If an architecture has only one hidden layer, the possibility of the number being three is great within all the possibilities. In the case of a selected random value failing to cause a

Algorithm 2: Crossover operator for PATF individuals

```
Data: patf, $n$, $k$//parents, amount, index
Result: childVet//new offspring
begin
    while $k \le (n/2)$ do
        $layers \leftarrow \mathbf{patf}(k).HL + \mathbf{patf}(k+1).HL$
        if $rand(1) > probs$ then
            $child.HL \leftarrow roundCeil(layers/2)$
        else
            $child.HL \leftarrow roundFloor(layers/2)$
        $dimensions \leftarrow [\mathbf{patf}(k).dim; \mathbf{patf}(k+1).dim]$
        $transFunc \leftarrow [\mathbf{patf}(k).func; \mathbf{patf}(k+1).func]$
        for $elements \leftarrow 1 : dimensions_{size}$ do
            $[child.dim, child.func] \leftarrow$
            $raffle(dimensions, transFunc)$
        $\mathbf{childVet} \leftarrow child$
        $k \leftarrow k + 2$
end
```

significant change in the number of hidden layers, a set of values between $arqval = [-2, 5]$ is generated and added to the current dimensions of the architecture. This range was defined empirically with the purpose of maintaining diversity among the individuals. If an architecture has two hidden layers, the possibility of the number being three is great, with the intention of reducing the number of hidden nodes per layer.

Algorithm 3: Mutation operator for PATF individuals

```
Data: childVet, $n$, $m$ //child vector, number of child, mutation rate
Result: mutChild
begin
    $childNumber \leftarrow ((m/100) * n)$
    while $childNumber > 0$ do
        $probability \leftarrow rand(1)$
        $child \leftarrow raffle(\mathbf{childVet})$
        if $child.numHiddLy = 1$ then
            if $probability \ge pInc$ then
                $child \leftarrow addLayers(3)$
            if $probability \ge pDec$ then
                $child \leftarrow addLayers(2)$
            else
                $child \leftarrow child + raffle(arqval)$

        else if $child.numHiddLy = 2$ then
            if $probability \ge pInc$ then
                $child \leftarrow addLayers(3)$
            if $probability \ge pDec$ then
                $child \leftarrow decreaseLayersTo(1)$
            else
                $child \leftarrow child + raffle(arqval)$

        else
            if $probability \ge pInc$ then
                $child \leftarrow decreaseLayersTo(2)$
            if $probability \ge pDec$ then
                $child \leftarrow decreaseLayersTo(1)$
            else
                $child \leftarrow child + raffle(arqval)$
        $\mathbf{childVet} \leftarrow \mathbf{childVet} - child;$
        $\mathbf{mutChild} \leftarrow child$
        $childNumber \leftarrow childNumber - 1$
    $\mathbf{mutChild} \leftarrow \mathbf{mutChild} + \mathbf{childVet}$
end
```

The evolutionary search for learning rules or learning algorithms occurs with the search for parameters from the following algorithms: Back-propagation (BP), Levenberg- Marquardt (LM), quasi-Newton Algorithm (QNA) and Scaled Conjugate Gradient (SCG). Based on the adopted learning algorithm, an individual from PLAP will have parameters from this algorithm for functioning. As a PLAP individual has a PATF, the fitness measure is given as the best I_{fit} among its set of architectures. The selection of the individual from PLAP also uses the tournament strategy. The mutation and elitism rates are the same presented earlier for the other evolutionary searches.

As individuals from PLAP have information related to the learning algorithm parameters in question, the crossover of individuals occurs with the generation of new values within a range based on the parent values. Based on probability, the mutation algorithm for the child performs an addition or subtraction operation in the child rates. Based on its current value, this operation causes a forty percent up or down (increasing or decreasing) change in the parameter.

The general functioning of the NNGA-DCOD method is presented in the Algorithm 4. Table I displays the complete list of the parameters used.

Algorithm 4: General functioning of the NNGA-DCOD

```
Data: Parameters in Table I
Result: A set of near-optimal ANNs
begin
    Generate randomly the populations of: initial weights, architectures and
    learning algorithms parameters
    foreach learning algorithm do
        Calculate the fitness for all individuals
        foreach $generation \in ESLAP$ do
            foreach
            $PATF \in PLAP$ and $generation \in ESATF$
            do
                foreach
                $PIW \in PATF$ and $generation \in ESW$
                do
                    Select parents to crossover based on the fitness
                    Apply genetic operators to produce the offspring for
                    next generation
                    Reduce the population to specified size
                Repeat steps from ESW
            Repeat steps from ESW
    Select the near-optimal networks found with every learning algorithm
end
```

The Evolutionary Search for Architectures and Transfer Functions (ESATF) is dependant on the Evolutionary Search for Weights (ESW). For every individual from PATF, there is one PIW; and for every population of initial weights, an ESW is performed for a fixed, pre-defined architecture. Therefore, the ESW occurs on a faster time scale than the ESATF. For every evolutionary search of architecture generation, many evolutionary searches of weight generations are performed. The Evolutionary Search for Learning Algorithm Parameters (ESLAP) occurs on a lesser time scale than the ESATF, because every individual from PLAP has one PATF that they evolve for many generations to find optimal architectures using pre-defined, fixed learning algorithm parameters. After one ESATF generation execution, for every individual from PLAP, there is one generation execution to search for learning algorithm parameters.

Parameters for:		Values
GAs	- Encoding	Direct
	- Elitism	10%
	- Mutation	40%
	- Selection	Tournament
	- Population/Generation	
	$\rightarrow$ Algorithms	7/30
	$\rightarrow$ Architectures	10/5
	$\rightarrow$ Weights	10/5
ANNs	- Type	MLP, feedforward
	- Transfer functions	Pure-linear (P), Tang-sigmoid (T), Log-Sigmoid (L)
	- Hidden layers	up to: 3
	- Hidden nodes	up to: 16
	- Training epochs	up to: 5
	- Range of initial weights	[-0.5, 0.5]
	- Output neuron	linear
Training algorithms	BP - Learning rate and momentum	[0.05, 0.25]
	LM - Learning rate	[0.001, 0.02]
	SCG - Step lengths	[1.0E-06, 100]
	- Limits on step sizes	[0.1, 0.6]
	QNA - Scale factor to determine performance	[0.001, 0.003]
	- Scale factor to determine step size	[0.001, 0.02]
	- Change in weight for second derivative approximation	[0, 0.0001]
	- Regulating the indefiniteness of the Hessian	[0, 1.0E-06]

TABLE I

NNGA-DCOD PARAMETERS.

The method starts the search by randomly generating populations for all kinds of individuals. Fitness is then calculated based on the ANN Normalized Mean Squared Error (NMSE) achieved in the training set. The genetic operators maintain the diversity of individuals for the tournament search of all layers and a small range of random selection, where the new individuals generated for the next offspring must be distinct from individuals in the present offspring. The amount of individuals used in the NNGA-DCOD is small, but as this method is iterative, with nested loops and, many new individuals are created at every generation of each kind of search. Thus, the search space explored is large and satisfactory results are achieved.

The improvements introduced in the NNGA-DCOD regard the change of the fitness calculation formula and change of the selection criterion. Currently, the measure of error is accomplished with the NMSE formula. Thus, the error information from the training set with NMSE is used for the calculation of the fitness of all individuals. The validation error is only used to present the gradual evolution of the search and the test error is only used to show the final performance of the near-optimal networks found. In the earlier versions of the NNGA-DCOD, the roulette wheel selection criterion was used, but according to experiments, this type of selection reduces population diversity, which is a phenomenon that strongly hampers the capacity of the GAs to find near-optimal solutions. A high diversity of individuals is something desired for GA applications [9], but as the diversity was not obtained through roulette wheel selection, tournament selection was adopted. Parameters from the genetic operators, such as the pressure rate of selection, were adjusted in order to accept networks with more hidden layers, less hidden nodes per layers and a satisfactory performance. Another improvement is the analysis of the time consumption in the search for near-optimal ANNs performed by the NNGA-DCOD and described together with the experimentation results in the next section.

III. EXPERIMENTATION SETUP

The experiments to evaluate the NNGA-DCOD was performed with five well-known classification problems found in the UCI repository [10]. Cancer with 9 attributes (atb), 699 examples (exp) and 2 classes (cla); Glass with 9 atb, 214 exp and 6 cla; Heart-Cleveland with 35 atb, 303 exp and 2 cla; Horse with 58 atb, 364 exp and 3 cla; and Pima-diabetes with 8 atb, 768 exp and 2 cla. To perform the experiments, we used five iterations of two-fold cross-validation (5 x 2 cv). At each iteration, data were randomly divided into halves. One half was the input for the algorithms (70% for training and 30% for the validation set) and the other half was used to test the final solution (test set). To determine whether the differences among the algorithms were statistically significant, we used a combined F-test described by [11]. Let $p_i^{(j)}$ denote the difference in the accuracy of two classifiers in fold j of the i-th iteration of 5 x 2 cv, $\bar{p} = (p_i^{(1)} + p_i^{(2)})/2$ denote the mean, and $s_i^2 = (p_i^{(1)} - \bar{p})^2 + (p_i^{(2)} - \bar{p})^2$ the variance.

$$f = \frac{\sum_{i=1}^{5} \sum_{j=1}^{2} \left(p_i^{(j)}\right)^2}{2 \sum_{i=1}^{5} s_i^2} \tag{6}$$

F is then approximately distributed with ten and five degrees of freedom. We rejected the null hypothesis that the two algorithms have the same error rate with a 0.05 significance level if $f > 4.74$. The accuracy results presented in the section 3.1 are based on error information from the ten tests and training sets. This methodology was used in experiments due to the fact that usual method generates an increase in type-I errors, where the results are incorrectly deemed significantly different more often than expected given the level of confidence used in the test [11].

The search for ANNs through trial-and-error was performed following the previously described methodology, using the same database split scheme and number of training epochs. We performed 10 runs in each fold for the following network setup: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22 and 24 hidden neurons for one hidden layer with the (T) transfer function. The purpose of these experiments was to compare the performance between NNGA-DCOD and the manual process, using the same database split scheme and number of training epochs.

A. Experimentation Results

Table II displays the mean values from near-optimal ANNs found with NNGA-DCOD and trial-and-error methods. The mean of the architectures (*arch*) is based on the amount of hidden units/neurons. Regarding the number of hidden neurons, the NNGA-DCOD found structurally better networks (with few hidden neurons) than those found with

Problems / Algorithms		NNGA-DCOD			Trial-and-error		
		Arch.	Errors		Arch.	Errors	
			Training	Test		Training	Test
Cancer	BP	12.8	6.26	6.22	16.8	23.16	23.14
	LM	3.5	1.85	2.52	11.4	1.91	2.67
	SCG	4.5	3.11	3.2	15	3.48	3.55
	QNA	11.7	3.14	3.22	7.4	3.39	3.53
Pima	BP	6.1	21.02	21.15	15.4	23.97	23.96
	LM	5.2	14.47	15.92	15	14.09	16.13
	SCG	2	16.3	16.9	12.4	18.13	18.51
	QNA	7.8	16.64	17.79	20.8	17.59	18.40
Heart	BP	12.1	14.33	14.26	21.2	24.62	24.64
	LM	5.4	7.02	12.25	16.4	6.48	13.14
	SCG	5.1	11.78	12.62	15.8	12.28	12.96
	QNA	13.3	12.6	12.49	17.8	12.04	12.89
Horse	BP	7.2	17.29	17.52	20	19.71	19.72
	LM	4.2	7.04	15.34	11.4	5.83	15.82
	SCG	4.6	13.52	15.28	12	12.96	15.51
	QNA	7.3	13.92	15.44	13.2	13.80	15.56
Glass	BP	5.6	12.73	12.75	11.7	13.93	13.94
	LM	12.3	7.61	9.18	14.6	7.89	9.62
	SCG	5.2	10.22	10.29	19	10.24	10.37
	QNA	5.1	10.47	10.53	12.6	10.34	10.44

TABLE II

MEANS VALUES FROM NEAR-OPTIMAL ANNS FOUND THROUGH NNGA-DCOD AND TRIAL-AND-ERROR.

the trial-and-error method for all problems. Moreover, with application of the f-test, the results of NNGA-DCOD also were statistically better than the manual method for all problems using the BP algorithm. In case of the Pima-Diabetes problem, the NNGA-DCOD found structurally better networks with the QNA algorithm as well.

In an overall analysis, NNGA-DCOD is very efficient in searching ANNs with very simple architectures and obtains a satisfactory error performance. In some cases, the developed method failed to outperform the trial-and-error method. This happened mainly when using algorithms that work with second-order information. These algorithms (LM, SCG and QNA) work with second derivative information and converge faster than first-order methods. Moreover, second-order algorithms do not experience as much interference from the variation in the amount hidden layers and neurons, type of transfer function and initialization weights as first-order algorithms do [3]. As NNGA-DCOD works with the variations in parameters to achieve better results (networks), good results are not always achieved using second-order algorithms. Analyzing the amount of hidden neurons, the ANNs found with NNGA-DCOD are much better than those found with the trial-and-error method and present a similar error performance for all problems. The capacity of NNGA-DCOD for searching ANNs with few hidden nodes is evident, but in the special case of the BP algorithm, the developed method achieved far better results in terms of errors and architectures in comparison to the manual method.

With the improvements to NNGA-DCOD, more complex neural networks were found and better error performances were achieved. The following are examples found using the SCG algorithm with the architecture configuration and test error: Cancer $1p$, $4t$, 2.42; Pima $1p$, $2t$, 15.89; Heart $3l$, $3p$, 12.25; Horse $2p$ $1t$, 14.12 and Glass $3t$, 10.43. Unlike previous versions of NNGA-DCOD, which found ANNs with only one hidden layer, the current version obtained networks with up to three hidden layers, which is considered near-optimal.

Table III displays the performance comparison between results obtained with NNGA-DCOD and other works found in the literature. The values in Table III are shown in same sequence as the problems discussed previously. This same sequence is valid for the means of nodes and connections. The methods used in the comparison do not work with learning algorithms for searching ANNs, as they having their own way to adjust the parameters of networks. Therefore, two versions of NNGA-DCOD were chosen in order to make the comparison more honest, in other words, the results with the BP (first-order) and SCG (which uses some second-order information) algorithms are used in the comparison.

Considering the figures in Table III which are mean values, NNGA-DCOD using BP is better than the other methods for searching near-optimal ANNs for the Horse and Glass problems regarding mean error and the amount of nodes. For the Pima-Diabetes and Heart problems, the version of NNGA-DCOD with SCG outperformed other methods with regard to the mean error and amount of nodes. In case of Cancer problem, the developed method achieved a satisfactory performance, placing it among the methods that present the best results for this problem.

Information on the amount of connections was considered in the comparisons, but NNGA-DCOD does not have special treatment for this ANN component. The other methods used in comparison have special mechanisms for connection eliminations to improve the performance of networks that are not trained with traditional algorithms (BP, LM, SCG or QNA). As the developed method does not have such a mechanism, comparisons are not appropriate, but were carried out with the intention of studying the power of the developed method. Especially for the Horse problem, NNGA-DCOD found networks with a small amount of connections. For the Heart problem, NNGA-DCOD found networks that needed a greater amount of nodes and, consequently, more connections in order to achieve good performances. The training algorithms also require a particular amount of nodes and connections in order to function successfully. For the Cancer, Pima-Diabetes and Glass problems, the mean amount of connections was similar to that of the other methods. Comparisons with other methods in the literature demonstrate that the developed method is very efficient in searching near-optimal ANNs.

The methods found in the literature on searching for near-optimal ANNs did not study the time consumption of search methodology. Table IV displays information on the time NNGA-DCOD needs to search for near-optimal ANNs for each problem using all training algorithms. Time is related to the search performed in one fold of the each problem. The main characteristic of methods that use evolutionary techniques (such as GAs) is the long processing time of the search. This is due to the fact that GAs considers a large search space (where near-optimal ANNs can be found) and,

Information	Methods						
	NNGA-DCOD with BP	NNGA-DCOD with SCG	GEPNET [6]	COVNET [6]	MOBNET [6]	COOPNN [5]	CNNDA^{ne} [7]
Errors	6.22	3.20	-	-	-	1.38	1.15
	21.15	16.90	19.27	19.90	19.84	21.35	19.91
	14.26	12.62	13.63	14.26	13.63	12.79	-
	17.52	15.28	-	-	-	26.37	-
	12.75	10.29	35.16	-	35.16	23.58	-
Nodes	12.8	4.5	-	-	-	5.89	3.5
	6.1	2	4.57	6.17	7.9	7.9	4.3
	12.1	5.1	6.37	4.77	11.4	7.28	-
	7.2	4.6	-	-	-	20.30	-
	5.6	5.2	6.33	-	14.87	6.73	-
Connect.	140.8	49.5	-	-	-	58.30	35.8
	61	20	24.60	29.43	76.32	76.32	40.4
	447.7	188.7	50.70	33.07	64.63	90.31	-
	439.2	280.6	-	-	-	779.67	-
	84	78	63.37	-	132.10	83.97	-

TABLE III
COMPARISON BETWEEN EVOLUTIONARY AND NON-EVOLUTIONARY^{ne} SEARCH METHODS.

Problems	Training Algorithms				Mean of Time
	BP	LM	SCG	QNA	HOURS
Cancer	26.10	30.77	36.15	27.86	**30.22**
Pima	26.84	30.90	34.72	27.55	**30.00**
Heart	26.44	38.51	46.20	26.58	**34.43**
Horse	28.57	80.44	39.12	28.35	**44.12**
Glass	27.68	32.38	29.96	27.53	**29.39**
Total mean	**27.13**	**42.60**	**37.23**	**27.58**	**33.63**

TABLE IV
MEAN TIME DISCRIMINATIONS IN HOURS OF PROCESSING TO SEARCHING NEAR-OPTIMAL ANNS.

consequently, requires a large amount of time to explore this search space [1].

This time requirement is the main disadvantage of the developed method and occurs because NNGA-DCOD has a search methodology with nesting loops for which there are currently no stopping criteria. Nonetheless, the figures in Table IV show interesting characteristics of the training algorithms. The greatest amount of processing time was expended with training algorithms that use second-order information. This can be explained by the fact that such algorithms require complex calculus to adjust the weights and, consequently, require more memory and processing time. The search with BP was faster because the training algorithm adjusts the weights in a simple way, requiring less memory and processing time. The search for ANNs in the Glass problem was faster due to the smaller amount of examples, whereas search was slower for the Horse problem due to the greater number of attributes.

IV. CONCLUSIONS

The search for ANNs that are custom tailored to a specific problem is considered a complex task and has mainly employed manual search methods. In this paper, we presented an improved version of a hybrid method for automatically searching near-optimal ANNs. This method is composed of a combination of GAs and ANNs that search different ANN information arranged in layers, in which the information searched for in every layer can be modified according a priori knowledge and the wishes of the user. Therefore, when there is satisfactory knowledge on the architectures, it is possible to restrict the search to a certain set of architectures, allowing a more extensive search for initial weights and learning algorithm parameters, for example. Another configuration of the framework adopted in this work can be made with the purpose of searching ANN information more intensively that may be unknown or little known to the user.

Improvements were made to the NNGA-DCOD and experiments were performed. The results demonstrate that this method is able to achieve compact neural networks with satisfactory performances when compared with a simulation of the manual search method. Furthermore, comparisons with other recent methods in the literature on searching for ANNs were carried out and the developed method achieved the best results regarding error performance and amount of hidden nodes. However, the NNGA-DCOD does not have a mechanism for dealing with connections; it works with fully-connected ANNs and does not find networks with few connections, whereas other methods have mechanisms for eliminating connections. Another disadvantage of the NNGA-DCOD is the considerable processing time of the search. Future work will be concentrated on reducing the complexity and processing time in the NNGA-DCOD as well as addressing the connection issues.

ACKNOWLEDGMENTS

The authors would like to thank CNPq (Brazilian Research Council) for their financial support.

REFERENCES

[1] K. P. Ferentinos, "Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms." *Neural Networks*, vol. 18, no. 7, pp. 934–950, 2005.

[2] A. Abraham, "Meta learning evolutionary artificial neural networks," *Neurocomputing*, no. 56, pp. 1–38, 2004.

[3] L. M. Almeida and T. B. Ludermir, "Automatically searching near-optimal artificial neural networks." *European Symposium on Artificial Neural Networks*, pp. 549–554, 2007.

[4] X. Yao, "Evolving artificial neural networks." *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[5] N. García-Pedrajas, C. Hervás-Martínez, and D. Ortiz-Boyer, "Cooperative coevolution of artificial neural network ensembles for pattern classification." *IEEE Trans. Evolut. Computation*, vol. 9, no. 3, pp. 271–302, 2005.

[6] N. García-Pedrajas, D. Ortiz-Boyer, and C. Hervás-Martínez, "Cooperative coevolution of generalized multi-layer perceptrons." *Neurocomputing*, vol. 56, pp. 257–283, 2004.

[7] M. M. Islam and K. Murase, "A new algorithm to design compact two-hidden-layer artificial neural networks." *Neural Networks*, vol. 14, no. 9, pp. 1265–1278, 2001.

[8] L. Ma and K. Khorasani, "New training strategies for constructive neural networks with application to regression problems." *Neural Networks*, vol. 17, no. 4, pp. 589–609, 2004.

[9] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[10] D. Newman, S. Hettich, C. Blake, and C. Merz, "UCI repository of machine learning databases," 1998. [Online]. Available: http://www.ics.uci.edu/ mlearn/MLRepository.html

[11] E. Cantú-Paz and C. Kamath, “An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems.” *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 35, no. 5, pp. 915–927, 2005.