Hybridization of evolutionary algorithms and local search by means of a clustering method

Alfonso Martínez-Estudillo, César Hervás-Martínez, Francisco Martínez-Estudillo, and Nicolás García-Pedrajas

Abstract

In this paper we present a hybrid evolutionary algorithm to solve nonlinear regression problems. Although evolutionary algorithms have proven their ability to explore large search spaces, they are comparatively inefficient in fine tuning the solution. This drawback is usually avoided by means of local optimization algorithms that are applied to the individuals of the population. The algorithms that use local optimization procedures are usually called hybrid algorithms. On the other hand, it is well known that the clustering process enables the creation of groups (clusters) with mutually close points that hopefully correspond to relevant regions of attraction. Then local search procedures can be started once in every such region.

This work proposes the combination of an evolutionary algorithm, a clustering process and a local search procedure to the evolutionary design of product-units neural networks. In the methodology presented, only a few individuals are subject to local optimization. Moreover, the local optimization algorithm is only applied at specific stages of the evolutionary process. Our results show a favorable performance when the regression method proposed is compared to other standard methods.

Keywords

Evolutionary algorithms, product-unit networks, clustering, hybridation.

I. Introduction

Evolutionary algorithms (EAs) are efficient at exploring an entire search space; however, they are relatively poor at finding the precise optimum solution in the region where the algorithm converges to. Many researchers [1] [2] [3] have shown that EAs perform well for global searching because they are capable of quickly finding and exploring promising regions in the search space, but they take a relatively long time to converge to a local optimum. During the last few years new methods have been developed in order to improve the lack of precision of the EAs using local optimization

This work has been financed in part by the TIC2002-04036-C05-02 project of the Spanish Inter-Ministerial Commission of Science and Technology (CICYT) and FEDER funds.

Alfonso Martínez-Estudillo is with the Department of Management and Quantitative Methods, ETEA, Spain. e-mail: acme@etea.com

C. Hervás-Martínez is with the Department of Computing and Numerical Analysis of the University of Córdoba, Spain. e-mail: chervas@uco.es

Francisco Martínez-Estudillo is with the Department of Management and Quantitative Methods, ETEA, Spain. e-mail: fjmestud@etea.com

Nicolás García-Pedrajas is with the Department of Computing and Numerical Analysis of the University of Córdoba, Spain. e-mail: npedrajas@uco.es

algorithms. These new methodologies are based on the combination of local optimization procedures, which are good at finding local optima (local exploiter), and evolutionary algorithms (global explorer). These are commonly known as hybrid algorithms.

On the other hand, clustering methods are a class of global optimization methods of which an important part include a cluster analysis technique. These methods create groups (clusters) of mutually close points that could correspond to relevant regions of attraction. Then, local search procedures can be started once in every such region, e.g. from its centroid. In order to identify such clusters, different methods of cluster analysis (also called pattern recognition, or automatic classification) are used [4].

Our approach proposes the combination of an evolutionary algorithm, a clustering process and a local search procedure for the evolutionary design of product-units neural networks for regression.

We incorporate a local search method in the evolutionary algorithm in order to improve its performance. But, if we want to efficiently use the hybrid algorithm, we have to reduce the computation time spent by the local search. In many problems, it is not efficient to carry out a local optimization algorithm for every individual in the population due to the size of the population and/or the dimension of the search space. So, our approach is to select a subset of the best individuals, perform a cluster analysis to group them, and optimize only the best individual of every group.

One of the main advantages of this method is that the computational cost of applying the local optimization algorithm to only a few individuals hardly affects the total time spent by the algorithm. The use of a clustering algorithm allows the selection of individuals representing different regions in the search space. In this way, the optimized individuals are more likely to converge towards different local optima. If we apply the optimization algorithm to every individual in the population we will obtain many similar solutions with a considerable waste of time.

The developed model is applied to the evolution of the structure and weights of product-unit based neural networks. This kind of network is a very interesting alternative to sigmoid-based neural networks, but they have a major drawback, in that the optimization algorithms usually employed for training typical feedforward neural networks are quite inefficient for training product-unit networks. So, an effective algorithm for training these networks is of great practical interest.

In order to test the performance of the proposed algorithms, the networks are applied to a benchmark regression problem and a hard real-world problem of estimation of the parameters of a second order model in microbial growth.

This paper is organized as follows: Section II provides a short review of related approaches; Section III describes our model in depth; Section IV is dedicated to a short description of product-unit based networks; Section V states the most relevant aspects of the evolution of product-unit neural networks using the proposed approach; Section VI explains the

experiments carried out; and finally Section VII summarizes the conclusions of our work.

II. Related work

There are several ways to use local optimization algorithms in an evolutionary algorithm. The particular combination is extremely important in terms of possible solution quality and computational efficiency. We need to find the right balance between local exploitation and global exploration. There are different combination strategies, for instance, multistart, Lamarckian learning, Baldwinian learning, partial Lamarckianism and random linkage [5] [6] [7].

In the multistart technique, several random starting points are generated and used as inputs into the local search and the best solution is recorded. This global optimization technique has been used extensively but is a blind search since it does not take into account past information [8]. The main disadvantage of the multistart method is that the local search may be initiated from points belonging to the same basin of attraction, leading to the discovery of the same minimum over and over, wasting computational resources. The clustering approach aims to form clusters of points that belong to the same region of attraction and then start a local search from only one point of each cluster. To form these clusters one starts from a uniform sample and then applies either a reduction [9], in which case a certain fraction of points with the highest function values is removed, or a concentration [10], where a small number of steepest descent steps are applied to every point in the sample, aiming to increase the sample density around the minima. Such methods, devised in [11] [12], paved the way toward the more efficient methods for Multilevel-Single Linkage (MLSL) [13] and Topographical MLSL [14] [15].

Incorporating an optimization algorithm into an evolutionary process gives rise to the concepts of Baldwin Effect and Lamarckian evolution. The Baldwin Effect allows an individuals fitness (phenotype) to be determined based on learning, that is, the application of local improvement [16], while Lamarckian evolution, in addition to using learning to determine an individuals fitness, changes the genetic structure of an individual to reflect the result of the learning [17]. Hybrid evolutionary algorithms need not be restricted to operating in either a pure Baldwinian or pure Lamarckian way. Instead, mixtures of both strategies, such as partial Lamarckianism [2] [7], could be employed.

In the context of recurrent neural networks, it is worth mentioning the work of Ku et al. [18]. The approach, named *low frequency of learning*, is a Lamarckian based evolutionary method that combines local search and cellular genetic algorithms. In order to reduce the computational cost, the local search algorithm is applied only at regular generation intervals.

From a different point of view, Kim and Cho [19] propose a hybrid genetic algorithm based on clustering to reduce the number of evaluations without any harmful effect on the performance of the algorithm. This approach does not

employ any form of local optimization, and there is no selection of the best individuals before applying the clustering technique. The algorithm is applied to the solution of optimization problems.

The Random linkage (RL) algorithm [8] is a search strategy for solving global optimization problems. The algorithm works by generating uniform random starting search locations, and applying accept/reject criteria to each of the generated locations. A combination of evolutionary algorithms with random linkage (EARL) is developed in an attempt to produce an evolutionary algorithm which does not repeat local searches by using RLs accept/reject criteria [2]. The above mentioned hybrid EAs have been used successfully to solve a wide variety of problems [7] [1] and experimental studies show that evolutionary algorithms and local search hybrids not only often find better solutions than simple EAs, but also may search more efficiently [20] [21] [22] [23]. A more detailed study of these hybrid methods and a comparison among them can be found in [24] [25].

III. Hybrid evolutionary programming algorithms

The proposed methodology is based on the combination of an evolutionary algorithm, a clustering process, and a local improvement procedure[1]. Among the different paradigms of Evolutionary Computation, we have chosen Evolutionary Programming (EP) due to the fact that we are evolving artificial neural networks. Although there is not a global consensus about the issue, most of the researches in this area agree that EP is the most suitable paradigm of Evolutionary Computation for evolving neural networks.

We have two different versions of the hybrid evolutionary algorithm depending on the stage when we carry out the local search and the cluster partitioning. The evolutionary algorithm without the clustering process and local search is represented by (EP). In the hybrid evolutionary programming (HEP) we run the EP without the local optimization algorithm and then apply it to the final best solution obtained by the EP. This allows the precise local optimum around the final solution to be found. The hybrid evolutionary programming with the clustering (HEPC) algorithm applies the clustering process on a large enough subset of the best individuals of the final population. In this method it is very important to determine the number of best individuals to consider as well as the number of clusters. After that, we apply the L-M algorithm to the best individual in each cluster. Finally, the algorithm named dynamic hybrid evolutionary programming with clustering (HEPCD) carries out both the clustering process and the L-M local search dynamically every G_o generations, where G_o must be fixed by the user. The final solution is the best individual among the local

[1]The local optimization algorithm used in our work is the Levenberg – Marquardt (L-M) optimization method [26] [27]. This algorithm is designed specifically for minimizing a sum-of-squares error [28]. Anyway, any other local optimization algorithm can be used in a particular problem.

optima found during the evolutionary process. These two methods, HEPC and HEPCD, are the ones that implement the ideas proposed in this paper. The other two methods, EP and HEP, are considered for comparison purposes.

The basic objective of our methodology is a reduction in the number of times a local optimization algorithm is applied. This is especially important when the local search algorithm is of a high computational cost. However, removing the local optimization procedure usually yields a worse performance. So, our approach is a good compromise, as we apply the optimization algorithm to a reduced number of individuals. Moreover, the clustering process avails us of the possibility of selecting a subset of individuals with different features. In this way, the optimization algorithm has a lower computational cost.

Another feature of our approach is that the optimized individuals are not included in the new population. Once the optimization algorithm is applied, we think that any further modification of the individual will be counter-productive.[2] So, these individuals are stored in a separate population till the end of the evolutionary algorithm.

As we have said, we have used a Levenberg-Marquardt algorithm as the local optimization method. As any gradient-based algorithm, the major drawback of L-M is that it can be trapped in a local optimum. In the evolution of product-unit neural networks, where small changes in weights can have a great impact on the output of the network, the error surface is extremely complex, and there is a high probability of being trapped in a local optimum. That is the reason why we apply the L-M algorithm after, at least, a partial evolution of the population.

IV. Product-unit neural networks

In order to test the validity of our model we have chosen a difficult problem, hard enough to justify the use of complex approaches. The problem is the automatic determination of the structure and weights of product-unit neural networks [29]. Product units enable a neural network to form higher-order combinations of inputs, having the advantages of increased information capacity and smaller network architectures when we have interaction between the input variables. Neurons with multiplicative responses can act as powerful computational elements in real neural networks [30]. Product-unit based networks are universal approximators [31].

Product unit based neural networks have a major drawback, since their training is more difficult than the training of standard sigmoid based networks. The backpropagation learning algorithm, the most common algorithm for training multi-layer neural networks, works best when the error surface is relatively smooth, with few local minima and plateaus. Unfortunately, the error surface for product units can be extremely convoluted, with numerous minima that trap

[2]Observe that we use the EP paradigm instead of Genetic Algorithms (GAs) or Genetic Programming (GP). In GAa or GP the inclusion of these optimized individuals would be advisable, as they may mate with other individuals to obtain even better offspring.

backpropagation. This is because small changes in the exponents can cause large changes in the total error. Several efforts have been made to develop learning methods for product units [32] [33] [34] [35] [36] [37] [38].

Let us consider the family of real functions F defined by:

$$F = \left\{ f : A \subset \boldsymbol{R}^k \to \boldsymbol{R} : f(\boldsymbol{x}, \boldsymbol{\theta}) = \beta_0 + \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} x_i^{w_{ji}} \right) \right\}, \tag{1}$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ is the input vector, $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m)$, $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_m)$, $\boldsymbol{w}_j = (w_{j1}, w_{j2}, \ldots, w_{jk})$, $\beta_j \in [-M, M] \subset \boldsymbol{R}$, $w_{ji} \in [-L, L] \subset \boldsymbol{R}$, $i = 1, 2, \ldots, k$, $j = 1, 2, \ldots, m$; $k, m \in \boldsymbol{N}$. The domain of definition of f is the subset A of $\boldsymbol{R}^k$ given by $A = \{(x_1, x_2, \ldots, x_k) \in \boldsymbol{R}^k : 0 < x_i \leq K_0\}$. Each function of the family can be seen as a polynomial expression of several variables, where the exponents of each variable are real numbers.

Every function of the family can be represented as a neural network. The network must have the following structure: an input layer with a node for every input variable, a hidden layer with several nodes, and an output layer with just one node. The nodes of a layer have no connections with each other, and there are no connections between the input and output layers. Figure 1 shows the structure of such a network.

The network has k inputs that represent the independent variables of the model, m nodes in the hidden layer, and one node in the output layer. The activation of j-th node in the hidden layer is given by:

$$B_j(\boldsymbol{x}, \boldsymbol{w}_j) = \prod_{i=1}^{k} x_i^{w_{ji}}, \tag{2}$$

where w_{ji} is the weight of the connection between input node i and hidden node j. The activation of the output node is given by:

$$\beta_0 + \sum_{j=1}^{m} \beta_j B_j(\boldsymbol{x}, \boldsymbol{w}_j), \tag{3}$$

where β_j is the weight of the connection between the hidden node j and the output node. The transfer function of all nodes is the identity function.

The local search algorithm is used for optimizing the weights of the neural network connections. The objective function used is the fitness value of the network.

V. Proposed model applied to product-unit neural network evolution

The evolution of product-unit networks uses the operations of selecting the best individuals of the population and two types of mutation: structural and parametric. We select the best $r\%$ individuals in the population, in our experiments

$r = 90$. There is no crossover, as this operation is usually regarded as less effective [39] for network evolution. In the following paragraphs we describe each one of the different aspects of the algorithm in detail.

Initial population. The algorithm begins with the random generation of a number of networks bigger than the number of networks used during the evolutionary process. We randomly generate $10N_P$ networks and we select the best N_P from among them.

So, we construct the initial population B whose size is N_P.

Selection plan. The $r\%$ best individuals of the population are selected from the set $B^* = B - \{B^{\text{best}}\}$ of cardinality $N_P^* = N_P - 1$, where B^{best} is the best individual of B. With these individuals we make up the population B' of size $\lfloor rN_P^*/100 \rfloor$.

Structural and parametric mutations. Every individual of the population B' is subject to structural mutation, obtaining B'_{struct}. The parametric mutation is applied only to the best $s = \lfloor (100-r)N_P^*/100 \rfloor$ individuals of B', obtaining B'_{param}. We construct the population $B'' = B'_{struct} \cup B'_{param}$ where the cardinality of B'' is $N_P^* = N_p - 1$.

Updated plan. The new population will be $B = B'' \cup \{B^{\text{best}}\}$ of cardinality N_P.

A. Clustering partitioning techniques and local search

Let $D = \{(\boldsymbol{x}_l, y_l) : l = 1, 2, \ldots, n_T\}$ be the training set, where the number of samples is n_T. We define the following application from the family of functions F to the Euclidean space $\boldsymbol{R}^{n_T}$:

$$\hat{\boldsymbol{y}}_f = (f(\boldsymbol{x}_l, \boldsymbol{\theta}))_{l=1,2,\ldots,n_T}\,, \tag{4}$$

where $\boldsymbol{\theta}$ is the set of parameters of f. The application assigns to each function of the family the vector obtained with the values of the function over the training data set. Then, we can define the distance between two functions of the family as the Euclidean distance between the associated vectors:

$$d(f,g) = \|\hat{\boldsymbol{y}}_f - \hat{\boldsymbol{y}}_g\| = \left(\sum_{l=1}^{n_T} |f(\boldsymbol{x}_l, \boldsymbol{\theta}) - g(\boldsymbol{x}_l, \boldsymbol{\theta})|^2\right)^{1/2}. \tag{5}$$

With this distance measure the proximity between two functions is related to their performance. So, similar functions will have similar performance over the same regression problem.

Now, considering a set of functions $\{f_1, f_2, \ldots, f_M\}$ of the family F, we can build the set of associated vectors $\{\hat{\boldsymbol{y}}_{f_1}, \hat{\boldsymbol{y}}_{f_2}, \ldots, \hat{\boldsymbol{y}}_{f_M}\}$ of $\boldsymbol{R}^{n_T}$. The minimum sum-of-squares clustering problem is to find a partition $P = \{C_1, C_2, \ldots, C_K\}$ of $\{\hat{\boldsymbol{y}}_{f_1}, \hat{\boldsymbol{y}}_{f_2}, \ldots, \hat{\boldsymbol{y}}_{f_M}\}$ in K disjoint subsets (clusters) such that the sum of squared distances from each point to the

centroid of its cluster is minimum. We use k-means clustering [40]. The election of the k-means algorithm has been made mainly because it is fast, simple and easy to implement. The reduction in the computational cost is one of the main objectives of our work, so a computationally heavy clustering algorithm would be counterproductive. In this algorithm, the cluster centroid is defined as the mean data vector averaged over all the items in the cluster. The problem can be expressed as

$$\min_{P \in P_K} \left\{ \sum_{j=1}^{K} \sum_{\hat{y}_l \in C_j} \|\hat{\boldsymbol{y}}_l - \bar{\boldsymbol{y}}_j\|^2 \right\}, \tag{6}$$

where $\bar{\boldsymbol{y}}_j = \frac{1}{N_j} \sum_{\hat{\boldsymbol{y}}_j \in C_j} \hat{\boldsymbol{y}}_j, N_j = |C_j|$, is the centroid of the j-th cluster and P_K denotes the set of all partitions of $\{\hat{\boldsymbol{y}}_{f_1}, \hat{\boldsymbol{y}}_{f_2}, \ldots, \hat{\boldsymbol{y}}_{f_M}\}$ in K sets. It is important to note that the centroid $\bar{\boldsymbol{y}}_j$ does not necessarily correspond to any concrete function of the population. We use the centroid only as a tool of the algorithm.

The number of clusters must be preassigned, and the initial partition is created randomly. Determining the number of clusters is not an easy problem because we may consider the merging of two classes into a single class or the splitting of a class into a number of classes.

The general structure of the hybrid algorithms is shown in Figures 2, 3, and 4. Figure 2 shows the hybrid evolutionary algorithm without clustering, Figures 3 and 4 depict the static and dynamic algorithms using clustering, respectively. As stated, we have three different algorithms depending on whether we use clustering and the stage when the clustering and local search algorithms are carried out:

1. We apply the L-M algorithm to the best individual obtained by the evolutionary algorithm in the final generation. Algorithm HEP (see Figure 2). The L-M algorithm optimizes the error function (7) with respect to the parameters $\boldsymbol{\theta}$ of the model.

2. We apply the clustering process to the best $\tilde{s}N_p$ individuals in the final population which is divided into K clusters $C_1, C_2, \ldots, C_K$. After that, we apply the L-M algorithm to the best individual of each cluster. The individuals obtained with the local-search algorithm are stored in a local optimum set C. Algorithm HEPC (see Figure 3). In this way, $\tilde{s}$ is the percentage of best individuals for clustering.

3. We apply the clustering process and then the L-M algorithm to the best individual of each cluster every G_o generations and in the final generation. The clustering process is applied to the best $\tilde{s}N_P$ individuals of the current population. The individuals obtained with the local-search algorithm are stored in C. Algorithm HEPCD (see Figure 4).

In cases 2 and 3, the final solution is the best individual among the local optima of set C. The evolutionary algorithm without clustering process and local search is represented by EP.

B. Fitness value

Let $D = \{(\boldsymbol{x}_l, y_l) : l = 1, \ldots, n_T\}$ be the training data set. The sum of squares error of an individual of the population that implements a function f is:

$$J = \frac{1}{2}\sum_{l=1}^{n_T}(y_l - f(\boldsymbol{x}_l))^2. \tag{7}$$

From this error we define the fitness function as:

$$A(f) = \frac{1}{1+J}. \tag{8}$$

C. Parametric and structural mutation

The structural mutation implies a modification in the structure of the function performed by the network and allows an exploration of different regions in the search space. The parametric mutation modifies the coefficients of the model using a self-adaptive annealing algorithm [41]. The severity of a mutation to an individual is dictated by the temperature, $T(f)$, given by:

$$T(f) = 1 - A(f), \qquad 0 \leq T(f) < 1. \tag{9}$$

Parametric mutation is accomplished for every coefficient w_{ji} and β_j of the network:

$$w_{ji}(t+1) = w_{ji}(t) + \varepsilon_1(t), \tag{10}$$

and

$$\beta_j(t+1) = \beta_j(t) + \varepsilon_2(t), \tag{11}$$

where $\varepsilon_k(t)$ represents a one-dimensional normally distributed random variable $\varepsilon_k(t) \in N(0, \alpha_k(t)T(f))$. The $\alpha_k(t)$ allow the adaptation of the learning process throughout the evolution following:

$$\alpha_k(t+1) = \begin{cases} (1+\lambda)\alpha_k(t) & \text{if } A(g_s) > A(g_{s-1}), \forall s \in \{t, t-1, \ldots, t-\rho\} \\ (1-\lambda)\alpha_k(t) & \text{if } A(g_s) = A(g_{s-1}), \forall s \in \{t, t-1, \ldots, t-\rho\} \\ \alpha_k(t), & \text{otherwise} \end{cases} \tag{12}$$

where $k \in \{1, 2\}$, $A(g_s)$ is the fitness of the best individual before the application of the local optimization algorithm, g_s, in generation s, λ and ρ must be set by the user. In our case we have considered $\alpha_1(0) = 0.1, \alpha_2(0) = 0.5, \lambda = 0.1$, and $\rho = 10$.

It should be pointed out that the modification of the exponents, w_{ji}, is different from the modification of the coefficients β_j, $\alpha_1(t) \ll \alpha_2(t), \forall t$. The parameters $\alpha_1(t)$ and $\alpha_2(t)$ determine, together with the temperature, the change in the variance of the distribution throughout the evolution, allowing the adaptation of the learning process. The adaptation of the parameters tries to avoid being trapped in local minima and to speed up the evolutionary process when the conditions of the searching are suitable. A generation is defined as successful if the best individual of the population is better than the best individual of the previous generation. If many successes are observed, this indicates that the best solutions are residing in a better region of the search space. In this case, we increase the strength hoping to find even better solutions closer to the optimum solution. If the fitness of the best individual is constant during several generations, we decrease the mutation rate. Otherwise, the mutation strength is constant.

There are five different structural mutations: Node addition (AN), node deletion (EN), connection addition (AC), connection deletion (EC), and node fusion (UN). The first four are identical to the mutations in the GNARL model [39]. In the node fusion, two randomly selected nodes, a and b, are replaced by a node c, which is a combination of the two. The connections whose source (outcoming) node is common to both the nodes under consideration are kept, with a weight given by:

$$\begin{aligned} \beta_c &= \beta_a + \beta_b \\ w_{ci} &= \frac{w_{ai} + w_{bi}}{2}. \end{aligned} \tag{13}$$

In this way the new node has a similar performance as the selected nodes, and the behavioral link is preserved as far as possible.

The connections that are not shared by the nodes are inherited by c with a probability of 0.5 and its weight is unchanged. For each mutation (excepting node fusion) there is a minimum value, Δ_{min}, and a maximum value, Δ_{max}, and the number of elements (nodes and connections) involved in the mutation is calculated as:

$$\Delta_{min} + \lfloor uT(f)(\Delta_{max} - \Delta_{min}) \rfloor, \tag{14}$$

where u is a random uniform variable in the interval $[0, 1]$. All the above mutations are made sequentially in the

given order, with probability $T(f)$, in the same generation on the same network. If the probability does not select any mutation, one of the mutations is chosen at random and applied to the network.

VI. Experiments

For each problem and each algorithm we have performed 30 runs with different random seeds. We have used the error measures *standard error prediction*:

$$SEP = 100 \times \frac{\sqrt{\frac{\sum_{i=1}^{N}(y_i-\hat{y}_i)^2}{N}}}{|\bar{y}|}, \tag{15}$$

and *mean squared error*:

$$MSE = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}. \tag{16}$$

where N is the number of patterns. We designate with MSE_G the mean squared error obtained over the generalization set.

In the rest of the paper we will use preferably the SEP for microbial growth problem, because it is non-dimensional and can be used to compare the error of dependent variables at different ranges and scales, and the MSE for the Friedman function.

A. Parameters of the algorithm

The exponents w_{ji} are initialized in the interval $[-5, 5]$, the coefficients β_j are initialized in $[-10, 10]$. The maximum number of nodes is $m = 6$. The size of the population is $N_P = 1000$. The stop criterion is reached whenever one of the following three conditions is fulfilled: i) The algorithm achieves a given fitness; ii) the values of $\alpha_1(t)$ and $\alpha_2(t)$ are below 10^{-4}; iii) for 20 generations there is no improvement either in the average performance of the 20% best individuals in the population or in the fitness of the best individual. For the Friedman function this stop criterion is modified, and the algorithm is always performed for 1800 generations.

The number of nodes that can be added or removed in a structural mutation is within the interval $[1, 2]$. The number of connections that can be added or removed in a structural mutation is within the interval $[1, 6]$. The only parameter of the L-M algorithm is the tolerance of the error to stop the algorithm, in our experiment this parameter has the value 0.01. The k-means algorithm is applied to $\tilde{s} = 25\%$ of the best individuals in the population. The number of clusters is 6 for both HEPC and HEPCD algorithms. For the latter algorithm $G_0 = 450$.

To adjust these parameters we have performed a study of the effect of the parameter values on the performance of the model. The study consisted of an analysis of variance using different values for the most important parameters: $\tilde{s}$, the percentage of the best individuals for clustering, K, number of clusters, and G, number of times we do cluster partitioning during the evolutionary process. Observe that $G_0 = (\text{Number of generations})/G$.

The main objective of the analysis is to determine if the influence of a change in a parameter value over the MSE_G is significant. We have performed the analysis for HEPCD algorithm. ANOVA examines the effects of some quantitative or qualitative variables (called factors) on one quantitative response, the MSE_G. The objective for that analysis is determining if the influence of a change in a parameter has a significant effect on the MSE_G obtained. Our linear model is of the form:

$$\text{MSE}_{Gijkl} = \mu + \tilde{s}_i + K_j + G_k + \tilde{s}K_{ij} + \tilde{s}G_{ik} + KG_{il} + \tilde{s}KG_{ijk} + e_{ikjl}, \tag{17}$$

where $i = 1, 2, 3$, $j = 1, 2, 3$, $k = 1, 2, 3$, $l = 1, \ldots, 10$, the first factor $\tilde{s}$ is the percentage of best individuals used and has levels $\{10\%, 25\%, 50\%\}$, K is the number of clusters in the last generation and has levels $\{2, 4, 6\}$, and G is the interval between clustering and has levels $\{2, 4, 6\}$. The $\tilde{s}K_{ij}$, $\tilde{s}G_{ik}$, and KG_{il} represent the interactions between each pair of factors, and $\tilde{s}KG_{ijk}$ and the interaction among the three factors. μ is a constant average term and e_{ikjl} represents a random noise. 270 experiments were carried out, corresponding to all the possible combinations of the three levels for each one of the three parameters. The ANOVA III results are shown in Table I.

The results show that there in no interaction between two of the three factors. The number of clusters has a significant effect on the value of MSE_G, with a best value of $K = 6$. The overall set of parameters with best results are $\{\tilde{s} = 25\%, K = 6, G = 4\}$ and $\{\tilde{s} = 50\%, K = 4, G = 4\}$. We choose the first set of parameters, that is less computationally expensive.

B. Friedman#1 function

This is a synthetic benchmark dataset proposed in [42] and widely used in the regression literature [43] [44]. Each instance of the training and test sets has 5 inputs. 1200 samples were created randomly, 200 for the training set and 1000, without noise, for the test set. The function is given by:

$$y = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \varepsilon, \tag{18}$$

where ε is a Gaussian random noise $N(0, 1)$ and the input variables are uniformly distributed in the interval $(0, 1]$. Table

II shows the results for Friedman#1 function. The results show that the dynamic approach obtains not only better results in mean, but also in the standard deviation.

Table III shows a comparison of the generalization results of the different proposed algorithms. The table shows the p-values of two-tailed t-tests comparing the means of the algorithms in pairs, as the normality test showed that the results of EP, HEP, HEPC, and HEPCD were normal. The tests show that HEPCD is the best performing algorithm, at a confidence level of 5% when compared with the other algorithms, and that HEPC is better than HEP at a confidence level of 10% and better than EP at a confidence level of 5%.

As the comparison tests show that the dynamic approach is the most suitable, we show the best model. This model achieves a MSE_G of 0.151 and has the following form:

$$\begin{aligned}
\hat{y} \quad = \quad & 6.221 + 123.710 x_1^{1.314} x_2^{1.499} \\
& -126.400 x_1^{1.641} x_2^{1.875} \\
& -58.334 x_3^{1.335} \\
& +58.255 x_2^{0.003} x_3^{1.695} \\
& +9.365 x_4^{1.073} \\
& +4.891 x_3^{0.021} x_5^{1.050}
\end{aligned} \tag{19}$$

These results can be compared with other works in the regression literature that use the same experimental setup. Vapnik [45] obtained a $\text{MSE}_G = 2.20$ using bagging techniques, $\text{MSE}_G = 1.65$ using boosting, and $\text{MSE}_G = 0.67$ using a Support Vector Machine. Roth [46] achieved a MSE_G over the original data set, before adding noise, of 2.84, 2.92, and 2.80 using Lasso, SVM and RVM methods respectively. Other works obtained similar results but with a slightly different experimental setup. Drucker [47] used 200 training samples and a test set of 5000 samples. Results for 10 runs with single trees, bagged trees and boosted trees produced a MSE_G of 4.65 for a single tree, 3.31 for bagging, 2.15 for boosting (with square loss), 2.79 for boosting (with exponential loss), and 2.84 for boosting (with square law loss). Granitto et al. [48] used neural network ensembles, with a training set of 300 patterns, and a test set of 1000 patterns. They used a random noise with distribution following a $N(0, 1)$. They also used a normalized mean-squared test error, that is, $\text{NMSE}_G = \text{MSE}_G/\sigma_D^2$, where σ_D^2 is the variance of the noisy data set. The results, in units of 10^{-2}, averaged over 50 experiments, and with a 20% of the data used for validation, are: 2.71 for a single network, 1.92 for bagging, 2.12 for epoch, 1.74 for SECA, and 1.80 for SimAnn.

C. Application to microbial growth

Acid Lactic bacteria (ALB) are considered the main microorganisms responsible for the deterioration of precooked packing meat products. These bacteria produce lactic acid, slime, and CO_2, which causes strange odors and tastes and affect the product acceptance [49]. In this work, we study the growth of the ALB microorganism Leuconostoc Mesenteroides, which has been frequently isolated as responsible for different alterations in meat products [50] [51].

To obtain absorbency data in Bioscreen C, a combination of factorial design and central composite design, CCD, was used to quantitatively assess the effects and interactions of the main factors that affect microbial stability in meat products: temperature T (^oC), pH, NaCl concentration (%), and $NaNO_2$ concentration (ppm). We wanted to study the combined effect of these factors in a culture medium *Tryptone Soy Broth (TSB)*. We carried out 7 experiments with every set of conditions. From the 7 experiments we randomly chose 5 experiments for the training set and the other 2 for the generalization set [52].

The collected data consist of 210 curves, representing the growth of the microorganism against time. These curves were adjusted to an exponential model [53] with the program DMFit 1.0 (Jï¿½sef Baranyi, Institute of Food Research, Norwich Research Park, Norwich NR4 7UA, UK). The models that describe the response of one or more kinetic parameters from the primary model are called secondary models. Typically there are three parameters that are interesting from the biologist's point of view: lag time, growth rate and y_{end}. These three values are shown on Figure 5 in a typical curve of microbial growth.

The growth rate (*grate*) is the maximum value of the growth curve slope. The lag-time (*lag*) is the instant in which the intersection between the line of the maximum slope and the lower asymptote of the growth curve is produced. y_{end} is the value of the asymptote of the growth curve. This dataset is available upon request to the authors.

The input variables of the problem have different scales and present a large range of variability. For this reason it is advisable to carry out a pre-processing of the data. We have done a simple linear rescaling to the input variables to the interval $[0.1, 1.1]$. The lower bound is chosen to avoid inputs values near 0 that could produce very large values of the function for negative exponents. The upper bound is chosen near 1 to avoid dramatic changes in the outputs of the network when there are weights with large values (specially in the exponents). The output variables are scaled in the interval $[1, 2]$.

Table IV shows the results for this problem. The table shows the values of SEP for learning and generalization.

As in the previous problem, the dynamic approach using clustering obtains the best results for all of the three problems. The two approaches using clustering outperform the other two models while also showing a better variance. This is

corroborated by the results of the t-tests shown in table V.

For this experiment we can obtain the lower bound of the SEP for the training set. This lower bound is given by the curve whose points are the marginal means of the statistical distribution of the training data (see [28] page 203).

If we obtain these values we have a SEP of 3.38 for the lag time, a SEP of 1.64 for the growth rate, and a SEP of 10.42 for y_{end}. We can see that the best values obtained by our hybrid model are close to these optimal values.

VII. Conclusions

In this paper we have proposed a new approach to solve nonlinear regression problems. This approach is based on the combination of an evolutionary algorithm, a clustering process and a local search procedure. The clustering process allows the selection of individuals representing different regions in the search space. These selected individuals are the ones subject to local optimization. In this way, the optimized individuals are more likely to converge towards different local optima.

We have proposed two different versions of the hybrid evolutionary algorithm depending on the stage when we carry out the local searches and the cluster partitioning. The results showed that the dynamic version obtains not only the better results in mean, but also in the standard deviation.

The hybrid evolutionary algorithm proposed was applied to two regression problems. In both of them, the approach is able to perform better than similar algorithms that do not use a clustering analysis. Although the computational cost is only slightly higher, the differences in accuracy/generalization performance between the proposed method and the method that does not use clustering is significant. This result suggests that the use of a clustering algorithm to select just a few individuals to optimize, instead of optimizing many of them, provides a very good compromise between performance and computational cost.

The experiments showed that the hybrid approach is able to obtain good results not only in a benchmark function, such as Friedman's, but also in a hard real-world problem, as the estimation of the parameters of a second order model in microbial growth.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees, whose valuable advise has contributed to greatly improve this paper.

References

[1] C. R. Houck, J. A. Joines, and M. G. Kay, "Comparison of genetic algorithms, random start, and two-opt switching for solving large location-allocation problems," *Computers and Operations Research*, vol. 23, no. 6, pp. 587–596, 1996.

[2] C. R. Houck, J. A. Joines, and M. G. Kay, "Empirical investigation of the benefits of partial lamarckianism," *Evolutionary Computation*, vol. 5, no. 1, pp. 31–60, 1997.

[3] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer–Verlag, New York, 1994.

[4] A. A. Torn and A. Zilinskas, "Global optimization," in *Lecture Notes in Computer Science 350*, pp. 255–263. Springer-Verlag, Berlin, 1987.

[5] A. Kollen and E. Pesch, "Genetic local search in combinatorial optimization," *Discrete Applied Mathematics and Combinatorial Operation Research and Computer Science*, vol. 48, pp. 273–284, 1994.

[6] N. L. J. Ulder, E. H. L. Aarts, H. J. Bandelt, P. J. M. Laarhoven, and E. Pesch, "Genetic local search algorithms for the travelling salesman problem," in *Parallel Problem Solving from Nature I*, H. P. Schwefel and R. Manner, Eds., Berlin, 1991, vol. 496 of *Lecture Notes in Compuer Science*, pp. 109–116, Springer-Verlag.

[7] J. A. Joines and M. G. Kay, "Utilizing hybrid genetic algorithms," in *Evolutionary Optimization*, R. Sarker, M. Mahamurdian, and X. Yao, Eds. Kluwer Academic Publisher, 2002.

[8] M. Locatelli and F. Schoen, "Random linkage: A family of acceptance/rejection algorithms for global optimization," *Mathematical Programming*, vol. 85, no. 2, pp. 379–396, 1999.

[9] R. W. Becker and G. V. Lago, "A global optimization algorithm," in *Proceedings of the 8th Allerton Conference on Circuits and Systems Theory*, 1970.

[10] A. A. Törn, "A search clustering approach to global optimization," in *Towards Global Optimization 2*, L. C. W. Dixon and G. P. Szegö, Eds. North-Holland, Amsterdam, 1978.

[11] C. G. E. Boender, A. H. G. R. Kan, G. T. Timmer, and L. Stougie, "A stochastic method for global optimization," *Mathematical Programming*, vol. 20, pp. 125–140, 1982.

[12] A. H. G. R. Kan an G. T. Timmer, "Stochastic global optimization methods. part i: Clustering methods," *Mathematical Programming*, vol. 39, no. 27–56, 1987.

[13] A. H. G. R. Kan an G. T. Timmer, "Stochastic global optimization methods. part ii: Multi level methods," *Mathematical Programming*, vol. 39, no. 57–78, 1987.

[14] A. A. Torn and S. Viitanen, "Topographical global optimization using pre sampled points," *Journal of Global Optimization*, vol. 5, pp. 267–276, 1994.

[15] M. M. Ali and C. Storey, "Topographical multilevel single linkage," *Journal of Global Optimization*, vol. 5, pp. 349–358, 1994.

[16] G. E. Hinton and S. J. Nolan, "How learning can guide evolution," *Complex Systems*, vol. 1, pp. 495–502, 1987.

[17] D. Whitley, S. Gordon, and K. Mathias, "Lamarckian evolution, the baldwin effect and function optimization," in *Parellel Problem Solving from Nature III*, Y. Davidor, H. P. Schwefel, and R. Manner, Eds. 1994, pp. 6–15, Springer-Verlag.

[18] K. W. C. Ku, M. W. Mak, and W-Ch. Siu, "A study of the lamarckian evolution of recurrent neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 1, pp. 31–42, April 2000.

[19] H-S. Kim and S-B. Cho, "An efficient genetic algorithm with less fitness evaluation by clustering," in *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, Seoul, Korea, 2001, pp. 887–894.

[20] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," Tech. Rep. 826, California Ins. Technol., Psadena, CA, 1989.

[21] P. Moscató and C. Cotta, "A gentle introduction to memetic algorithms," in *Handbook of Metaheuristics*, F. Glower and G. Kochenberger, Eds., pp. 1–56. Kluwer, 1999.

[22] H. Bersini and B. Renders, "Hybridizing genetic algorithms which hill-climbing methods for global optimization: Two possible ways," in *Proc. IEEE Int. Symp. Evolutionary Computation*, Orlando, FL, 1994, pp. 312–317.

[23] C. J. Merz, "A principal components approach to combining regression estimates," *Machine Learning*, vol. 36, no. 1, pp. 9–32, July 1999.

[24] J. He, J. Xu, and X. Yao, "Solving equations by hybrid evolutionary computation techniques," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 295–304, September 2000.

[25] R. Sarker, M. Mohammadian, and X. Yao, Eds., *Evolutionary Optimization*, Kluwer Academic Publishers, 2002.

[26] K. Levenberg, "A method for solution of certain non-linear problems in least squares," *Quarterly Journal of Applied Mathematics II*, vol. 2, pp. 164–168, 1944.

[27] D. W. Marquardt, "An algorithm for least-squares estimation of non-linear parameters," *Journal of the Society of Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[28] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[29] R. Durbin and D. Rumelhart, "Product units: A computationally powerful and biologically plausible extension to backpropagation networks," *Neural Computation*, vol. 1, pp. 133–142, 1989.

[30] E. Salinas and L. F. Abbott, "A model of multiplicative neural responses in parietal cortex," *Proc. National Academic of Sciences USA*, vol. 93, pp. 11956 – 11961, 1996.

[31] M. Leshno, V. L. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, pp. 861–867, 1993.

[32] D. J. Janson and J. F. Frenzel, "Training product unit neural networks with genetic algorithms," *IEEE Expert*, vol. 8, no. 5, pp. 26–33, October 1993.

[33] A. Ismail and A. P. Engelbrecht, "Training product units in feedforward neural networks using particle swarm optimisation," in *Development and Practice of Artificial Intelligence Techniques, Proceeding of the International Conference on Artificial Intelligence*, V. B. Bajic and D. Sha, Eds., Durban, South Africa, 1999, pp. 36–40.

[34] L. R. Leerink, C. L. Giles, B. G. Horne, and M. A. Jabri, "Learning with product units," in *Advances in Neural Information Processing Systems 7*, pp. 537–544. MIT Press, 1995.

[35] M. Schmith, "On the complexity of computing and learning with multiplicative neural networks," *Neural Computation*, vol. 14, pp. 241–301, 2001.

[36] K. Saito and R. Nakano, "Law discovery using neural networks," in *Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI '97)*, 1997, pp. 1078–1083.

[37] K. Saito and R. Nakano, "Extracting regression rules from neural networks," *Neural Networks*, vol. 15, no. 10, pp. 1279–1288, 2002.

[38] E. Oost, S. ten Hagen, and F. Schulze, "Extracting multivariate power functions from complex data sets," in *Proceedings of the 14th Dutch-Belgian Artificial Intelligence Conference, BNAIC'02*, Leuven, Belgium, 2002.

[39] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54 – 65, January 1994.

[40] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 1990.

[41] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[42] J. Friedman, "Multivariate adaptive regression splines (with discussion)," *Ann. Stat.*, vol. 19, pp. 1–41, 1991.

[43] J. Carney and P. Cunningham, "Tuning diversity in bagged ensembles," *Int. J. Neural Systems*, vol. 10, no. 4, pp. 267–279, 2000.

[44] W. M. Lee, Ch. P. Lim, K. K. Yuen, and S. M. Lo, "A hybrid neural network model for noisy data regression," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 34, no. 2, pp. 951–960, 2004.

[45] V. Vapnik, *The nature of Statistical Learning Theory*, Springer, 1999.

[46] V. Roth, "The generalized lasso," *IEEE Transactions on Neural Networks*, vol. 15, no. 1, 2004.

[47] H. Drucker, "Improving regressors using boosting techniques," in *Proceedings of the Fourteenth International Conference on Machine Learning*, D. H. Fisher Jr., Ed., San Mateo, CA, 1997, pp. 107–115, Morgan Kaufmann.

[48] P. M. Granitto, P. F. Verdes, and H. A. Ceccatto, "Neural network ensembles: Evaluation of aggregation algorithms," *Artificial Intelligence*, vol. 163, pp. 139–162, 2005.

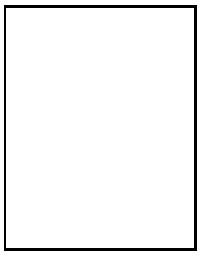
[49] J. H. J. Huis in't Veld, "Microbial and biochemical spoilage of foods: An overview," *Int. J. Food Microbiology*, vol. 33, pp. 1–18, 1996.

[50] P. Mäkelä, *Lactic acid bacterial contamination at meat processing plants*, Ph.D. thesis, College of Veterinary Medicine, Helsinki, 1993.

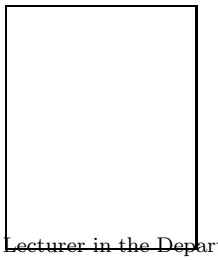
[51] K. Björkroth and H. Korkeala, "Evaluation of lactobacillus sake contamination in vacuum packaged sliced cooked meat products by ribotyping," *J. Food Prot.*, vol. 59, no. 398–401, 1996.

[52] R. M. García, C. Hervás, M. R. Rodríguez, and G. Zurera, "Modelling the growth of leuconostoc mesenteroides by means of an artificial neural network," *International Journal of Food Microbiology*, 2005, in press.

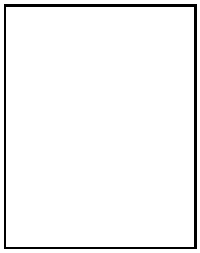
[53] J. Baranyi and T. A. Roberts, "A dynamic approach to predicting bacterial growth in food," *Int. J. Food Microbiol.*, vol. 23, pp. 277–294, 1994.



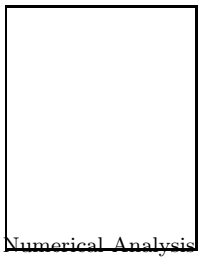
Alfonso C. Martínez Estudillo Alfonso C. Martínez Estudillo received his B.S. degree in Computer Science in 1995, and his Ph.D. in 2005, both from the University of Granada, Spain. He is currently a Lecturer in the Department of Management and Quantitative Methods in ETEA, University of Córdoba, Spain. His current areas of interest include neural networks, evolutionary computation and regression.



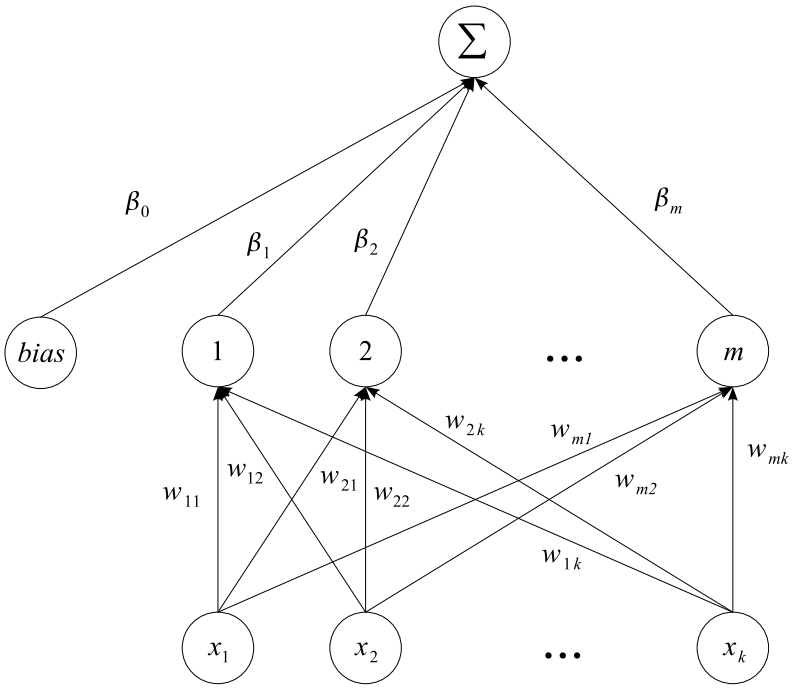
César Hervás-Martínez César Hervás-Martínez was born in Cuenca (Spain). He received his B.S. degree in Statistics and Operating Research in 1978 from the Universidad Complutense of Madrid (Spain). He received his Ph.D. degree in Mathematics at the University of Seville in 1986. His current research interests include neural networks, evolutionary computation, and modeling of natural systems. He is an Associate Professor at the University of Córdoba in the Department of Computing and Numerical Analysis in the Area of Computer Science and Artificial Intelligence and Lecturer in the Department of Quantitative Methods in the School of Economics. He is a member of the IEEE and other technical societies.



Francisco Martínez Estudillo Francisco J. Martínez Estudillo received the B.S. degree in Mathematics in 1987, the Ph. D. degree in Mathematics in 1991, Differential Geometry speciality, both from the University of Granada, Spain. From 1987 to 2002, he developed his research in non-Euclidean geometry, Lorentz spaces and maximal surfaces. He is currently Associate Professor in the Department of Management and Quantitative Methods in ETEA, University of Córdoba, Spain. His current areas of interest include neural networks, evolutionary computation and regression.



Nicolás García-Pedrajas Nicolás García-Pedrajas was born in Córdoba (Spain) in 1970. He received his B.S. degree in Computing in 1993 from the University of Málaga (Spain). He received his Ph.D. at the University of Málaga in 2001. His current research interests include neural networks, evolutionary computation and game playing. His dissertation research involves the automatic design of artificial neural networks using both genetic algorithms and evolutionary programming. He is an Associate Professor at the University of Córdoba in the Department of Computing and Numerical Analysis in the Area of Computer Science and Artificial Intelligence. He is a member of several technical societies including the ACM, the INNS, the IEEE and the IEEE Computer Society.


∑
Outputlayer
β_0
β_1
β_2
β_m
bias
1
2
...
m
Hidden layer
w_{2k}
w_{m1}
w_{mk}
w_{11}
w_{12}
w_{21}
w_{22}
w_{m2}
w_{1k}
x_1
x_2
...
x_k
Input layer

1. Generate initial population $B(0)$ of solutions
2. while stopping criteria not met do
3. Select $B' \subset B(t)$
4. Apply structural mutation to every individual of B'
   to produce B'_{struct}
5. Apply parametric mutation to the best s individuals of B'
   to produce B'_{param}
6. $B'' = B'_{struct} \cup B'_{param}$
7. $B(t+1) = B'' \cup \{B^{\text{best}}\}$
8. $t = t + 1$
9. end while
10. Apply L-M local search to the best individual obtained
11. Return the best individual obtained

1. Generate initial population $B(0)$ of solutions
2. while stopping criteria not met do
3. Select $B' \subset B(t)$
4. Apply structural mutation to every individual of B'

   to produce B'_{struct}
5. Apply parametric mutation to the best s individuals of B'

   to produce B'_{param}
6. $B'' = B'_{struct} \cup B'_{param}$
7. $B(t+1) = B'' \cup \{B(t)^{\text{best}}\}$
8. $t = t + 1$
9. end while
10. Apply clustering process to a subset of $B(\infty)$, the population of the last generation,

    and L-M local search and save in C
11. Return the best individual of C

1. Generate initial population $B(0)$ of solutions
2. while stopping criteria not met do
3. Select $B' \subset B(t)$
4. Apply structural mutation to every individual of B'

   to produce B'_{struct}
5. Apply parametric mutation to the best s individuals of B'

   to produce B'_{param}
6. $B'' = B'_{struct} \cup B'_{param}$
7. $B(t+1) = B'' \cup \{B(t)^{\text{best}}\}$
8. Apply clustering to a subset of $B(t+1)$ and L-M local search every G_o generations and save in C
9. $t = t + 1$
10. end while
11. Apply clustering process to a subset of $B(\infty)$, the population of the last generation,

    and L-M local search and save in C
12. Return the best individual of C

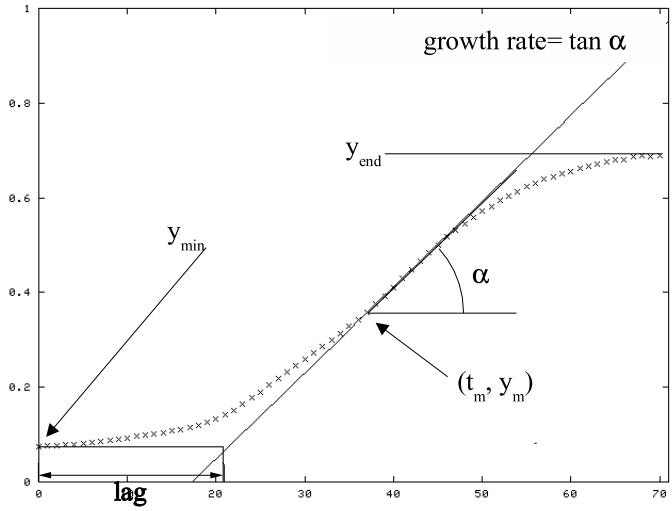

growth rate= tan α
y_end
y_min
α
(t_m, y_m)
lag
0
0.2
0.4
0.6
0.8
1
0
20
30
40
50
60
70

Fig. 1. Model of a product unit based neural network.

Fig. 2. HEP algorithm

Fig. 3. HEPC Algorithm

Fig. 4. HEPCD Algorithm

Fig. 5. Representation of lag time, growth rate and y_{end}

Source	SS	DF	MS	F	p-value
$\tilde{s}$	0.013	2	0.006	0.025	0.975
K	1.926	2	0.963	3.746	**0.025**
G	0.106	2	0.053	0.206	0.814
Residual	67.613	263	0.257		

TABLE I

ANOVA RESULTS FOR THE MSE_G (RESPONSE) WITH THREE PARAMETERS AS FACTORS. THE RESULTS WITH A SIGNIFICANCE LEVEL OVER THE 95% ARE IN BOLDFACE. SS REPRESENTS SUM OF SQUARES, DF DEGREES OF FREEDOM, MS , AND F IS THE VALUE OF THE SNEDECOR'S F.

Algorithm	Learning				Generalization				#Evaluations
	Mean	Std. dev.	Best	Worst	Mean	Std. dev.	Best	Worst	
EP	1.443	0.255	1.212	2.183	0.827	0.366	0.398	1.632	1205848.85
HEP	1.124	0.145	1.002	1.470	0.276	0.196	0.152	0.958	1206028.75
HEPC	1.099	0.124	1.002	1.470	0.244	0.130	0.152	0.581	1206730.15
HEPCD	1.082	0.106	0.994	1.351	0.234	0.116	0.151	0.521	1210255.35

TABLE II

RESULTS OF MSE FOR FRIEDMAN#1 FUNCTION FOR 30 RUNS WITH NETWORKS OF A MAXIMUM OF 6 HIDDEN UNITS, $\tilde{s} = 25\%$, $K = 6$, AND $G = 4$.

	HEP	HEPC	HEPCD
EP	0.000	0.000	0.000
HEP	–	0.094	0.030
HEPC	–	–	0.025

TABLE III

p-VALUES OF THE t-TEST FOR THE GENERALIZATION RESULTS OF FRIEDMAN #1 FUNCTION .

		Learning				Generalization			
Parameter	Algorithm	Mean	Std. dev.	Best	Worst	Mean	Std. dev.	Best	Worst
LnLag	EP	4.774	0.445	4.005	5.685	5.683	0.284	5.146	6.408
(4:5:1)	HEP	4.232	0.436	3.650	5.240	5.519	0.200	5.024	5.792
	HEPC	4.055	0.347	3.616	5.207	5.460	0.220	5.024	6.017
	HEPCD	3.977	0.291	3.616	5.207	5.316	0.151	4.975	5.555
Grate	EP	4.356	1.764	3.075	9.870	4.459	1.790	3.207	10.205
(4:5:1)	HEP	3.682	1.600	2.036	8.151	3.937	1.625	2.637	8.809
	HEPC	3.101	1.340	1.840	7.252	3.429	1.226	2.510	7.689
	HEPCD	2.803	0.938	1.840	5.856	3.068	0.730	2.510	5.848
y_{end}	EP	15.719	1.190	14.348	20.714	20.243	0.998	18.807	21.995
(4:6.1)	HEP	14.998	1.148	13.926	19.849	19.717	0.748	18.504	21.168
	HEPC	14.582	0.819	13.920	18.103	19.560	0.576	18.610	20.736
	HEPCD	14.311	0.385	13.920	15.297	18.937	0.673	16.973	19.905

TABLE IV

Results for microbial growth for 30 runs. Between parenthesis the structure of the evolved networks.

LnLag

	HEP	HEPC	HEPCD
EP	0.002	0.000	0.000
HEP	–	0.048	0.000
HEPC	–	–	0.001

Grate

	HEP	HEPC	HEPCD
EP	0.000	0.000	0.000
HEP	–	0.000	0.000
HEPC	–	–	0.012

y_{end}

	HEP	HEPC	HEPCD
EP	0.000	0.000	0.000
HEP	–	0.134	0.000
HEPC	–	–	0.042

TABLE V

p-VALUES OF THE t-TESTS FOR THE PARAMETERS OF THE SECOND-ORDER MODEL OF MICROBIAL GROWTH