Available online at www.sciencedirect.com
ScienceDirect
Information Sciences 177 (2007) 2884–2905
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters

P.A. Castillo [a,*], J.J. Merelo [a], M.G. Arenas [b], G. Romero [a]

[a] Departamento de Arquitectura y Tecnología de Computadoras, ETSI Informática, Universidad de Granada, Spain
[b] Departamento de Informática de la Universidad de Jaén, Escuela Politécnica Superior, Universidad de Jaén, Avda. Madrid, 35, E. 23071 Jaén, Spain

Received 1 February 2006; received in revised form 14 February 2007; accepted 14 February 2007

Abstract

In this paper we present a comparative study of several methods that combine evolutionary algorithms and local search to optimize multilayer perceptrons: A method that optimizes the architecture and initial weights of multilayer perceptrons; another that searches for training algorithm parameters, and finally, a co-evolutionary algorithm, introduced here, that handles the architecture, the network's initial weights and the training algorithm parameters. Our aim is to determine how the co-evolutive method can obtain better results from the point of view of running time and classification ability. Experimental results show that the co-evolutionary method obtains similar or better results than the other approaches, requiring far less training epochs and thus, reducing running time.

Keywords: Evolutionary algorithms; Artificial neural networks; Multilayer perceptrons; Hybrid algorithms; Optimization; G-Prop; Co-evolution

1. Introduction

Designing an artificial neural network (ANN) is not an easy process, as has been pointed out, for instance, by Duprat et al. [29]. It requires setting a layer structure with connections among the different components, tuning several parameters (such as initial weights) and the defining a set of learning constants. Subsequently, a training method is used, which is usually an iterative gradient descent algorithm designed to minimize, step by step, the difference between the actual output vector and the desired one. For multilayer perceptrons (MLPs), algorithms such as backpropagation (BP), QuickProp (QP) [33] or RPROP [95] are commonly used.

* Corresponding author. Tel.: +34 958240589.
E-mail address: pedro@atc.ugr.es (P.A. Castillo).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2007.02.021

However, in practice, gradient descent methods, successful as they are in many fields, do encounter certain difficulties: (i) the convergence to the global optimum tends to be extremely slow and is not guaranteed, and (ii) learning constants and other parameters, such as layer size must be found heuristically.

These problems have traditionally been approached using various optimization procedures, such as incremental/decremental methods [1,86,85,87,88,76,102] or evolutionary algorithms [103,16]. *Incremental* algorithms add hidden neurons to the smallest possible network until the required precision is reached. They start off with a few hidden neurons which increase in number until the error is small enough. The problem is that once the hidden neurons have been added, they cannot be suppressed to reduce the network size, thus resulting in unwieldy ANNs. In general, as Hwang et al. proved in [54], adding new units leads to overfitting. On the other hand, *decremental* algorithms start off with a network with a high number of connections and nodes and remove them one by one (or set them to zero) to obtain a smaller network with the same (or better) classification ability. Normally, the ANN must be re-trained after pruning. If the selection criteria (the units to be discarded and the order in which they will be removed) are guessed correctly, it is possible to obtain a good solution. The main issue with incremental/decremental approaches is that they are also gradient descent optimization methods, so they may reach the local minimum closest to the search space point where the process started [2].

There are other parameters besides layer size: due to their influence in the neural network performance [57,18], both the learning parameter and the number of training epochs (i.e. training set presentations) must be established. A small value of the learning parameter may slow down the convergence and trap the gradient descent algorithm in the local minima, while a value higher than the required could result in an overstepping of the optimum. Rumelhart et al. [97] have proposed suitable values for this constant, which would allow the algorithm to resolve a wide range of problems. On the other hand, a low number of training epochs might not be enough to learn the training set, whereas a number of training epochs higher than the one required may result in overfitting.

Overfitting can be avoided by taking into account the generalization error and stopping the training [82] if the minimum has been detected, although it is not easy to establish the point at which training has to be stopped. Generalization ability may vary during the training process, to the extent that at a given point, the generalization error reaches a minimum, after which it starts to grow [94] when the network learns some characteristics that are not representative of the training set. Thus, early detection of overfitting occurring during supervised training is important in order to prevent the error from increasing. In this sense, experiments carried out by Keesing and Stork [57] show the importance of the number of training epochs. In a given population of ANNs, applying either very little or too much training leads to a very slow evolution, the optimal usually being an intermediate amount.

One effective way to resolve overfitting and parameter setting is the use of evolutionary algorithms (EAs) [103,105]. However, as the optimization of all possible parameters would result in too large a search space, some authors propose the use of co-evolution [83,53].

Summing up, designing a neural classifier from scratch involves searching in the architecture and learning parameters space for a solution that is optimal from the point of view of its classification ability.

In this paper, we take the research already carried out on evolutionary optimization of MLPs (*G-Prop* method) presented in [10,15,18] a step further. G-Prop is a hybrid algorithm that leverages the capabilities of two classes of algorithms: the ability of the EA to find a solution close to the global optimum, and the ability of the QP algorithm to tune a solution and reach the nearest local minimum by means of a local search performed from the solution found by the EA.

This method will be compared with other hybrid approaches to optimize the MLP, with respect to the classification ability, reducing the number of training epochs at the same time: as the training time decreases, the required time to design an MLP with good classification ability will decrease too. Hybrid methods compared in this paper are the following:

- *G-Prop* (see Section 3.1), an EA designed to optimize MLPs. This method optimizes the neural network structure, establishing the number of hidden units and initial weights.
- *Sequential QP evolution* (see Section 3.2), an EA that searches for the optimal QP algorithm parameter values: number of training epochs and learning constant value.

- *Co-evolutive* (see Section 3.3), introduced in this paper, where a population of QP evolves in parallel with a population of MLP. Both populations cooperate to optimize the network architecture (number of hidden units and connection weights) and the training parameters (learning coefficient and number of training epochs) to solve a classification problem. We expect to obtain good generalization error, while reducing the running time.

We intend to show the aspects where the co-evolutive method yields better results, compared to other methods.

The rest of this paper is structured as follows: Section 2 presents the state of the art in co-evolutionary algorithms for neural network optimization. Section 3 describes the proposed methods. Section 4 describes the experiments, and Section 5 presents the results obtained, followed by a brief conclusion in Section 6.

2. Related research

EAs [75,32] are global optimization methods, based on the theory of evolution of natural species and its molecular basis [24]. EAs carry out a multidirectional search by maintaining a population of potential solutions, creating and interchanging information in all directions within the search space. The population undergoes simulated evolution: In every generation the best solutions mate and get copied, while the worst disappear. In order to distinguish between good and bad solutions, an objective function (evaluation or fitness function) that measures the quality of the solution is used.

Evolutionary neural networks are an efficient way of searching the problem space of the neural net [103,3,22]. The initial approach consisted in restricting the search space to a few parameters. Thus, several authors have suggested evolving ANNs by coding the weights and learning parameters of the individuals of EA, pre-establishing the number of neurons and the connectivity between them [47,49,71,90,48,64]. However, these representations can lead to a lack of precision by restricting the search to just a part of the possible space. Leung et al. [64] proposed the tuning of the parameters of a neural network using an improved genetic algorithm (GA), but the number of hidden nodes must be chosen manually by increasing it from a small number until the learning performance is good enough. The method proposed in [68] tries to avoid overfitting by encoding the number of training epochs as a bit string in the individual chromosome. Another approach based on simulated annealing that searches for the network learning parameters was proposed by the authors in [14].

Such methods focus on optimizing a part of the ANN, since the optimization of all possible parameters would result in too large a search space. For this reason, some authors propose the use of co-evolution (the mutual evolutionary influence between several species) [83,53], whereby several populations evolve in different EAs. According to dependencies between species, interactions between populations of different classes of individuals, the following classification could be made:

- Competitive co-evolutionary algorithms [96,81]: The fitness of an individual depends on how it performs in competition with individuals from other species (each species competes with the remainder).
- Cooperative co-evolutionary algorithms [91]: The fitness of an individual depends on its ability to cooperate with individuals from other species in order to solve a target problem.

In those cases where the problem can be divided into interrelated subproblems, a cooperative model is suitable [91].

Many authors have successfully used co-evolution in conjunction with different methods [21]. Historically, Hillis [51] was the first person to propose the optimization of sorting networks using a cooperative co-evolutionary algorithm. One species represents sorting networks and the other tests cases by sorting number. The interaction between the two species adopts the form of complementary fitness functions: a network is evaluated on how well it sorts test cases, while the test cases are evaluated on how poorly they are sorted.

Several authors have proposed the use of cooperative models to design only the network architecture. One example is the work of Moriarty and Miikkulainen [79,80] who developed a method for designing ANNs based on two EAs: a population of nodes and another of networks (different ways of combining nodes of the first population). The nodes are coded by using floating point vectors to represent weights. A population

of networks evolves records of neurons that work well together. The fitness of a neuron is computed as the average fitness of the best networks it participates in.

A more complex system was proposed by Zhao in [106]. His method breaks down a pattern classification problem into several functions (one per class) and assigns a module-network to each function. Thus, the whole classifier (network) consists of N sub-systems, and these sub-systems (module-networks) are in turn explored by evolution using several EAs. In [107], he addresses the problem by focusing on the evaluation of the modules which form the whole classifier, applying a cooperative co-evolutionary model to the evolutionary design of radial basis function ANNs.

Similar cooperative methods try to develop subnetworks (modules) instead of whole networks [91,66,40,41,43]. These modules, which must cooperate, are combined forming ensembles which make up a network. The fitness assignment is based on both competition within species and cooperation among species. The method proposed by Chandra and Yao [19,20] uses a multi-objective evolutionary algorithm for the construction of neural ensembles. It tries to find an optimal trade-off between diversity and accuracy and it searches for an ensemble for some particular problem by treating these two objectives separately.

Hallinan and Jackway [50] propose a cooperative feature selection algorithm which utilizes a genetic algorithm to select a feature subset in conjunction with the weights of an ANN. Each network is encoded as a single binary string, whereby every eight bits represents either a feature or a weight. This type of codification may result in a loss of precision, and good solutions could be lost due to its limitations.

The co-evolution of multiple cooperative species has been applied to job-shop scheduling [52] and a dual-species cooperative model has also been applied to Goldberg's three-bit deceptive function [84]. These species have a symbiotic relationship whereby the second species used the representations co-evolved by the first species.

Other authors present non-cooperative methods. Smalz and Conrad [98], for example, propose the use of two separately evolved populations: a population of nodes, divided into clusters, and a population of networks which are combinations of neurons, one from each cluster. This method neither induces competition among the neurons of the same cluster nor enforces cooperation among the different neuron clusters.

Nearly all of the methods previously discussed concentrate on optimizing the network structure and ignore the importance of training parameters. We have detected that not only should the network architecture and weights be optimized, but also the training parameters through a co-evolutionary method.

3. Hybrid methods

We have studied different hybrid systems for ANN optimization in the present document. In general, hybrid systems, combine global with local search methods (for instance, Lee and Lee in [62] combine genetic algorithms and ant colony optimization algorithms with local heuristics to improve the final result). Methods described in the following subsections are designed to search not only for the network architecture but also for the training algorithm parameters.

3.1. Single-population method for MLP optimization (G-Prop)

In G-Prop, an EA carries out the evolution of an MLP population. It searches for the best architecture (network structure and initial weights) for that problem and tries to optimize the network classification ability. This method makes use of the capabilities of two types of algorithms: the ability of EA to find a solution close to the global optimum, and the ability of the QP algorithm to tune it and to reach the nearest local minimum by means of local search from the solution found by the EA.

The complete description of the method and the results obtained using classification problems have been presented elsewhere [9–13,15,17,18]. The designed method uses an elitist [100] algorithm and is specified in Fig. 1.

In G-Prop, an individual is a complete MLP with two hidden layers. The representation ability of a neural network depends on the number of layers, on the number of neurons per layer and on the connectivity between layers. It was demonstrated that a network with two hidden layers can solve any pattern classification

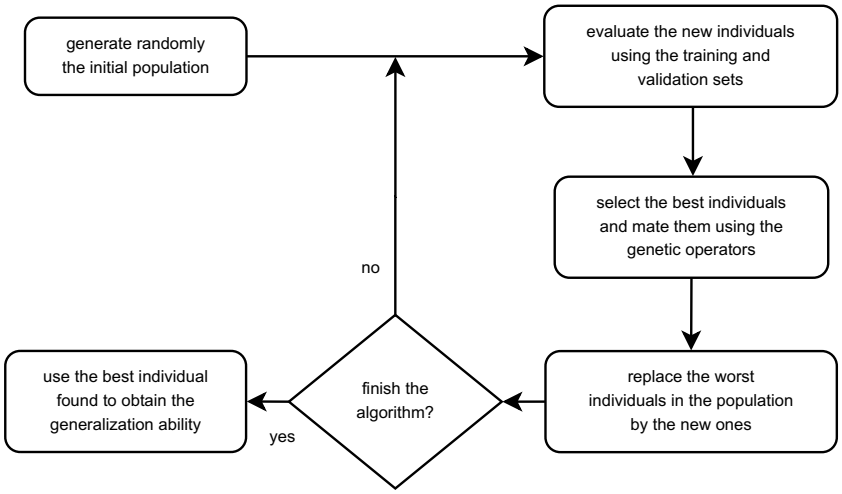


Fig. 1. EA pseudocode.

problem [65,6,94]. On the other hand, several authors have proved that any function approximation problem can be solved by using one hidden layer [60,38,37,23].

An EA requires that each individual is encoded as a chromosome for it to be handled by the genetic operators of the EA. Some authors use binary or real encoding (representation of the networks in a binary or real number string), as proposed by de Falco et al. [26] and Durr et al. [30], or indirect coding, as proposed by Cangelosi et al. [7] and Gruau [48], but G-Prop evolves the initial parameters of the network (initial weights and learning constants) using specific genetic operators. At the lowest level, an MLP is an object instantiated from the MLP C++ class. The data structure of this class is an array of vectors of neurons, where each neuron is a vector of weights. However, the EA does not use binary strings, but MLP objects and neurons.

The genetic operators act directly upon the ANN object (instead of performing hierarchical evolution at the neuron level [78]), but only *initial weights* and the *learning constant* are subject to evolution, not the weights obtained after training. In order to calculate the fitness, a clone of the MLP is created, and thus, the initial weights remain unchanged in the original MLP. Only when the training operator is used, changes are saved back into the individual genetic code that remains in the population.

When a genetic operator changes an MLP, it considers each hidden neuron (and its input and output weights) as a "gene", so that if two MLPs are crossed, complete hidden layer neurons are interchanged (and weights to and from it are treated as one unit), as proposed in [99,71,70].

Six *genetic operators* were designed to evolve MLPs:

- The *mutation operator* randomly changes the weights of certain neurons (see Fig. 2), depending on the application percentage. This operator resembles that of Montana and Davis [77] and the algorithm presented by Kinnebrock in [59]. Modifications consist of changing the *in and out* weights of a neuron by adding a small random number uniformly distributed in the interval $[-0.1, 0.1]$. Mutation also affects the learning rate, which is modified by adding a small random number that follows *uniform* distribution in the interval $[-0.05, 0.05]$.

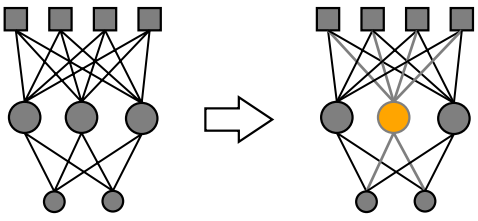


Fig. 2. The mutation operator randomly changes the *in and out* weights of certain neurons.

- The *crossover operator* performs two point crossover between two MLPs to produce two new MLPs. The hidden layer neurons are a mixture of those of the two parents: some of the hidden neurons, together with the *in* and *out* connections from each parent, make up one offspring, and the remaining hidden neurons comprise the other (see Fig. 3). Finally, the learning rate is swapped between the two nets.
- The *training operator* is used to improve the individual MLP through a local search (applying QP), as suggested by Montana and Davis [77] and Yao and Xu [104], who proposed applying backpropagation (BP) as a variation operator that tunes an ANN. When applied, the operator takes an MLP that is trained for a specified number of epochs, and returns it, with the trained weights, to the population.
- The *operator that adds hidden neurons* and the following one (*elimination*) attempt to solve one of the main problems of BP and its variants: the difficulty of guessing the number of hidden layer neurons (see Fig. 4). By means of the addition of hidden neurons, it is no longer necessary to set the size of the EA search space. This operator is intended to perform incremental design: it starts with a small structure and increments it, if necessary, by adding new hidden units. Now, however, the dilemma of overfitting arises: small networks generalize well, but are slow at learning, whereas big networks learn fast (needing fewer training epochs to obtain similar precision), but generalize badly [5,4]. The way to obtain good generalization ability is to use the smallest network that can learn the input data efficiently [93,94,103]. For this reason, this operator is combined with the following one. In addition, it is balanced by the component of the fitness function that penalizes the network size.
- *Removing hidden neurons operator* is intended to perform decremental ANN design: it prunes certain nodes and, if this is performed properly, obtains better generalization results and a smaller network [56,89,4]. This means, to a certain extent, that the networks are prevented from growing too much and reaching an excessive size.
- *Substitution operator* replaces one hidden layer neuron at random with a new one, initialized with random weights. This operator may be considered a kind of macromutation that affects only one gene. Macromutation can destroy whatever the net learned, but can act as a jump in the search space to other areas, so that the MLP can fall in a good one and, by means of another operator of refinement can obtain a good solution.

The *fitness function* of an individual (MLP) is given by the number of correctly classified patterns obtained on the validation process that follows training. In the case of two individuals showing an identical classification error (measured as the number of incorrectly classified patterns), the one with the hidden layer containing

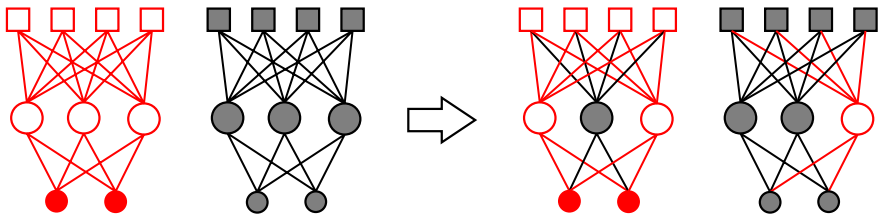


Fig. 3. The crossover operator takes two MLPs. Some of the hidden neurons make up one offspring and the remaining hidden neurons comprise the other.

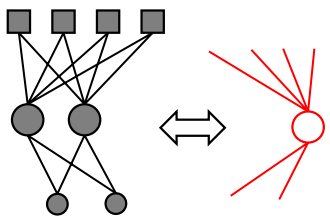


Fig. 4. The operator that adds hidden neurons inserts a new randomly initialized neuron. The removing hidden neurons operator prunes certain nodes together with the *in and out* connections.

the least number of neurons would be considered the best (the aim being small networks with a high generalization ability).

G-Prop does not need to set by hand any of the MLP parameters (except the number of training epochs to evaluate the population individuals, since, depending on the problem, a higher or lower value will be necessary). Obviously, the EA constants (some of them concerning the initial population, and others concerning the genetic operators) need to be set. Influence of these parameters was analyzed in a previous work [18] where the most suitable values for each parameter were determined. The methodology used and the values obtained are detailed in Section 4.3.

3.2. Single-population method for QP optimization (evQP)

This method evolves the QP parameters, trying to reduce the number of epochs in order to minimize running time. If the number of epochs is reduced, the fitness computation is faster, and the global running time decreases. The designed algorithm follows an elitist schema [100].

This EA works on a population of individuals, coding the number of epochs and the learning coefficient used to apply the QP algorithm to train an MLP. The chromosome is composed of an integer number which codes the number of training epochs and of a floating point number that codes the initial learning coefficient value.

The evolution of this kind of individual is made by means of the application of two genetic operators:

- *Mutation operator* changes the number of training epochs, adding a random number uniformly distributed in the interval $[-10, 10]$, and the learning constant by adding a small random number uniformly distributed in the interval $[-0.01, 0.01]$.
- *Crossover operator* exchanges the number of training epochs and the learning constant between two individuals.

At startup, the method randomly generates 10 MLPs with random weight values in a specified range, and random hidden layer sizes (ranging between 2 and 90). Those MLPs are not changed during the run. To obtain the fitness of an individual, the parameters being coded by this individual are used to train those 10 MLPs (the validation error is obtained and the average value is used as fitness value). This method does not use any operator to change the network size.

The average classification ability obtained is used as the first criterion of the fitness. In the case of two individuals showing an identical classification error, the one with a smaller number of training epochs will be considered the best.

3.3. Co-evolutive method for MLP and QP optimization

This option is inspired by the method proposed by Potter and De Jong in [91]: a system where two species evolve in a cooperative way, and we are introducing it for the first time in this paper. The architecture is an ecosystem of two species, which are genetically isolated by simply evolving them in separate populations. The species interact with each other through their fitness functions and maintain a cooperative relationship (Fig. 5). Thus, proposed model consists of two processes that are executed in parallel:

- One *EA evolves a population of QP algorithms that optimizes the training algorithm parameters* (number of training epochs and initial learning parameter). This algorithm is identical to the single-population method used to optimize QPs, described in Section 3.2. However, to evaluate an individual (QP), several MLPs taken from the other population are used.
- The second *EA evolves a population of MLPs to optimize both the network architecture and the initial weights*. This EA optimizes the MLP classification ability and simultaneously searches for the network architecture (number of hidden units) and the initial set of weights. This algorithm is identical to the method described in Section 3.1, although in order to calculate the MLP fitness, a QP taken from the other population is used to train the network.

a

MLP population

(1) Generate the initial population with MLPs with random weight values in a specified range, and random hidden layer sizes.

(2) Repeat for M generations:

(a) Evaluate the new MLPs (individuals): train them using the training set and obtain their fitness according to the number of correct classifications on the validation set and the network size. The number of epochs and learning coefficient are obtained from an individual-QP sent from the other population.

(b) Select the S best individuals in the population, according to their fitness, to mate using the genetic operators to obtain the new individuals.

(c) Replace the S worst individuals in the population by the new ones.

QP population

(1) Generate the initial population of QPs. The number of epochs and initial learning parameter are initialized with random values.

(2) Repeat for G generations:

(a) Evaluate the new individuals: train MLPs from the other population, using the number of training epochs and learning parameter that the individual-QP codes, using the training set and obtaining their fitness according to the number of correct classfications on the validation set.

(b) Select the N best individuals in the population, according to their fitness, and apply the genetic operators to obtain the new individuals.

(c) Replace the N worst individuals in the population by the new ones.

b

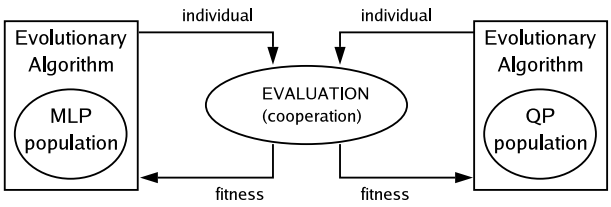


Fig. 5. Proposed co-evolutive method to optimize MLPs and QPs: (a) algorithm pseudocode and (b) method schema. Both EAs run in separate processes, evolving the MLP population and the QP population in parallel, which in turn cooperate in designing the MLP and in setting the learning parameters.

In each generation, the EA that evolves MLPs needs to evaluate the offspring. The MLP evaluation consists of the network training for a number of times (using the training set) and using a learning constant value. The method G-Prop (described in Section 3.1) requires that these parameters are manually established before the EA is run. Nevertheless, in this third method, a second EA searches for the number of training epochs and the learning constant value. Thus, these training parameters are obtained from a non-evaluated QP-individual sent from the second population.

At the same time, the second EA has generated several individual-QP in the previous generation. In order to evaluate them, the training parameter values codified by it are taken and used to train some MLPs. Instead of using a fixed set of MLPs (as in the method described in Section 3.2), several MLPs from the first EA are trained. Both the MLPs and the QPs obtain their fitness values at the same time:

- The MLPs obtain their fitness value as the classification error on the validation set. If two individuals have identical classification errors, the best will be the one that has the hidden layer with the least neurons (see the first method described above, Section 3.1).
- The fitness value assigned to the individual-QP uses as a first criterion the average MLP classification ability. The second decisive factor will be the number of training epochs required to minimize the running time (see the second method described above, Section 3.2).

Through cooperative evaluation of QPs and MLPs, we intend to guarantee that the optimized parameters from evolutionary population of QP algorithms are suitable to be applied on evolved population of MLPs.

3.4. Program implementation and test-bed

In the co-evolutive method, communication implementation between processes was carried out using the Perl module `SOAP::Lite` [61]. SOAP is a standard protocol proposed by the W3C that extends the remote procedure call to allow remote access to objects. It is also a high-level, lightweight protocol, simple and extensible, used in application communication. EAs were implemented using the `Algorithm::Evolutionary` module [69,73], available at `http://opeal.sourceforge.net` and `http://search.cpan.org/author/JMERELO/`.

Experiments were run on several machines, whose speed ranged between 700 and 1200 MHz. The machines were connected using a 100Mbit non-dedicated Ethernet network.

4. Experiments

The tests used to assess the accuracy (obtained error) of a method must be carefully selected, since some synthetic problems (also called "toy problems") are not suitable for certain capacities of the BP algorithm, such as generalization [34]. We agree with the view put forward by Prechelt [92], stating that two real problems should be used in order to test an algorithm.

In real life problems, the division between classes is not as clear as it is in synthetic problems. The dispersion of samples within a single class is also greater, due to noise [72,74]. In any case, the best way to test the algorithm ability as well as its limitations is to use it to resolve real world problems.

In this paper, Glass and Breast Cancer pattern classification problems were used. These problems were put forward by Prechelt in his paper "*PROBEN1 – A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms*" [92]. We deal with two very different problems, regarding both the total number of patterns in each dataset and the number of patterns in each class.

Prechelt [92] proposed three different partitions out of the total set of patterns in the dataset, forming the training, validation, and test sets. These three different partitions present different difficulties, so that an algorithm might work better on a partition and worse on another one. This is in agreement with the *No Free Lunch* theorem [101], according to which there is no algorithm better than all to solve all the problems. In these experiments, the partitions proposed by Prechelt which are available at `http://page.mi.fu-berlin.de/~prechelt/NIPS_bench.html` were used.

4.1. Glass

The dataset is based on the glass problem dataset from the UCI library of machine learning databases. This task was prompted by the needs of forensic scientists involved in criminal investigation. The results of a chemical analysis of glass splinters (content of eight different elements in percentage terms) together with a refractive index, are used in the classification of the sample as either float-processed or non-float-processed building windows, vehicle windows, containers, tableware or head lamps. It contains 214 entries. Each sample has nine attributes plus the class attribute (type of glass): refractive index, sodium, magnesium, aluminium, silicon, potassium, calcium, barium, and iron.

This dataset is very difficult to classify due to two important features. First, the number of available patterns is low (214) for six different classes. Second, the number of patterns in each class is very unbalanced, ranging from 76 (building windows non-float processed) to 9 (tableware).

4.2. Breast cancer

This dataset comes from the UCI machine learning dataset "Wisconsin breast cancer database", which was compiled from the University of Wisconsin Hospitals and Clinics in Madison by Dr. William H. Wolberg [67]. Prechelt gives an exhaustive report [92] on this dataset, among others. Each sample has 10 attributes plus the class attribute: sample code number, clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, mitoses, class (0 for

benign, 1 for malignant). The class distribution in the original set is as follows: 65.5% benign and 34.5% malignant.

4.3. Methodology

Datasets were divided into three disjoint parts: one for training, one for validating, and one for testing, as proposed in [92]. In order to determine the fitness of an individual, the MLP was trained by the training set and its fitness was established from the classification error with the validating set. After the EA has finished, i.e. when it has reached the limit of generations, we obtain the generalization ability by using the testing set (previously unseen patterns). This generalization value is represented as arrays.

Our aim is to avoid an MLP learning the training set, gets a high fitness value and it is the best at the end of the run. Later on, it could be unable to classify correctly not previously seen patterns. The fitness assignment consists of two steps: (1) training the network which is using the training set, (2) using the validation set (not previously seen patterns) to obtain the classification capacity. Even though an MLP specializes and learns the validation set, it has to classify not previously seen patterns in order to obtain the generalization ability (shown in tables).

On the other hand, genetic algorithm users adjust the main design parameters of an EA (crossover probability, mutation probability, population size, number of generations, selection rate) by hand [25,55]. The decision on which values are optimal is usually made in terms of the most common values or experimental formulae given in the bibliography, or by trial and error [58,45]. Nevertheless, it is very important to know which parameter values involved in the design of an EA have the greatest influence on its behavior and performance. When making a detailed statistical analysis of the influence of each parameter, the designer should pay greater attention to the parameter providing the values that are statistically most significant.

Some other authors carried out a statistical study [18] in order to determine the most important parameters (regarding their influence on the results), and to establish the most suitable values for such parameters (thus obtaining an optimal operation). In this study, the ANOVA (ANalysis Of the VAriance) [35,36] statistical method was used. This statistical tool, based on the analysis of the mean variance, is widely used.

The ANOVA method was used to determine whether a change in the responses is due to a change in a factor or to a random effect. Besides the ANOVA method, the statistical analysis tool ANOM (ANalysis Of Mean) was used. This technique uses the average values for each parameter level (value) and the main effect (on the responses) plots, to decide the most suitable values for each parameter. As a result, running parameter values shown in Table 1 were obtained.

The number of generations and the population size needed for greater diversity should, of course, be higher or lower depending on the difficulty of the problem.

Table 1
Parameters set using statistical methods (see [19] for details)

Parameter	Value
Number of generations	500
Population size	500
Selection rate	20%
Initial weights range	$[-0.05, 0.05]$
Mutation operator priority	2.0
Crossover operator priority	0.5
Addition operator priority	1.0
Elimination operator priority	0.5
Training operator priority	0.5
Mutation probability	0.4
Weight mutation range	$[-0.001, 0.001]$
Learning constant mutation range	$[-0.010, 0.010]$

Since the co-evolutive method is based on G-Prop, results obtained in [18] have been used to establish the parameter values. However, to avoid long runs, experiments were set using from 100 up to 600 generations and a population size of 100 individuals.

The first method (see Section 3.1) used 300 training epochs and 0.1 as the learning coefficient, while co-evolutive and sequential methods to optimize QPs also search for these parameters (values ranged between 80 and 240 for training epochs; and between 0.01 and 0.7 for learning constant).

To conclude these experiments, the results of the methods described in this paper will be compared with the following "simpler methods" (always taking into account that computational cost must be the same for all experiments):

- In the experiments reported here, a large number of networks were trained (up to 600 generations of 100 individuals, i.e. 60,000 networks). Thus, it would be interesting to see if the use of evolutionary techniques presents any advantage over, for instance, a simple enumeration of a number of possible settings. The aim of including this method (*QP alone*) is to compare the proposed hybrid methods with a number of previously selected settings. We have chosen 300 different options for the network architecture and 200 options for the parameter settings, which leads to a total of 60,000 options. Those values were chosen following a uniform distribution. The generalization error shown below has been obtained using the testing set.
- An interesting approach would be to test whether the two evolutionary algorithms are independent. In this case, another experiment would involve applying the first and second EA sequentially versus using co-evolution (we would evolve the MLP parameters and the QP -*GProp* + *evQP*-). This method first runs the *G-Prop* program and then, the *evQP* program. In this way, we would be able to determine whether the MLP evolution is sufficiently independent from QP parameter evolution process, combining these two techniques and testing whether the results are better than those obtained by just applying either of them.

Time was measured using the Unix time command. The operating system calculated the running time (the real time elapsed between the start and the finish, the user CPU time, and the system CPU time measured in minutes) from which mean and standard deviations (for 30 runs) shown in tables were obtained (see Section 5).

Statistical *t-Student* tests are used to evaluate obtained results and to test whether differences among means are significant.

5. Results

Three tables are shown for each problem. They show the generalization performance on previously unseen patterns, the running time and the number of training epochs obtained using the other methods (EA to optimize MLPs -*GProp*-; the sequential method used to optimize QP parameters -*evQP*-; and the *co-evolutive* method). The results were also compared with the experiments carried out using simpler methods (applying *QP alone*; and evolving the QP parameters prior to evolving MLP -*GProp* + *evQP*-).

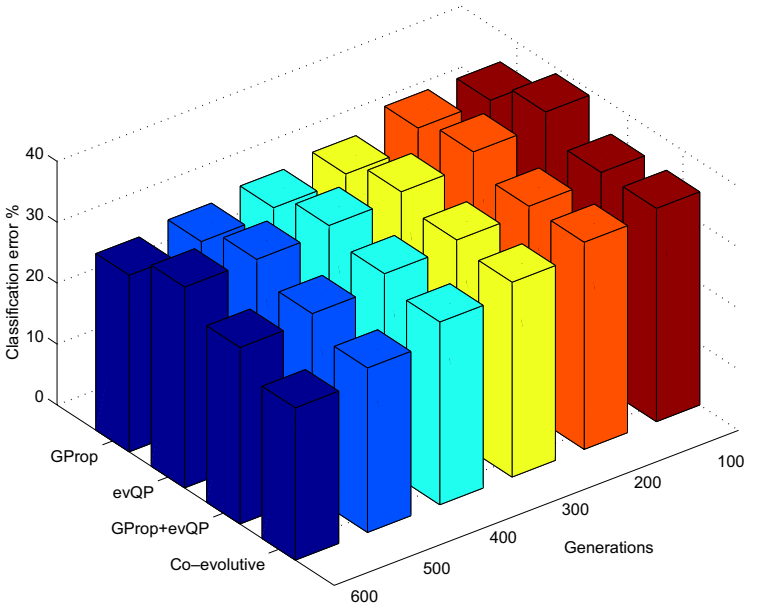
Results shown in tables are obtained as the mean and standard deviation (as commented above), and the experiment setup is such that the computational cost is the same for all proposed models.

Sections 5.1 and 5.2 present results obtained using the first partition of datasets, proposed by Prechelt [92] and used by Grönroos [46]. Section 5.3 presents results obtained using the second and third dataset partition. It tries to verify the conclusions obtained after the analysis carried out in previous subsections.

5.1. Glass-I

Results obtained on the Glass-I partition showing classification error, running time, and number of training epochs can be found in Tables 2–4. The best results have been highlighted in order to ease interpretation. In some cases, asterisks have been used to indicate significative differences regarding the other results.

Table 2
Generalization error rate obtained after running the experiments (Glass-I)



Generations	GProp	evQP	GProp + evQP	Co-evolutive
100	**35 ± 1**	39 ± 3	35 ± 1	**35 ± 2**
200	35 ± 1	37 ± 1	**34 ± 3**	**34 ± 3**
300	**32 ± 2**	35 ± 3	33 ± 2	**32 ± 2**
400	31 ± 2	34 ± 2	32 ± 3	**30 ± 3***
500	30 ± 4	33 ± 4	30 ± 3	**27 ± 4***
600	29 ± 3	33 ± 1	29 ± 3	**25 ± 3***

The best results are highlighted in order to ease interpretation (asterisks indicate significant differences regarding the other results). Plot shows how the classification error is reduced with the number of generations. However, the evQP method obtains worse results, while the co-evolutive method outperforms the other methods.

As can be seen, *G-Prop* increases its classification ability with the number of generations, but running time also increases. Results on error obtained using *evQP* are slightly worse due to the fact that this method does not optimize either the network architecture or the initial weights, but only focuses on the optimization of training parameters (as the number of training epochs decreases, so does running time).

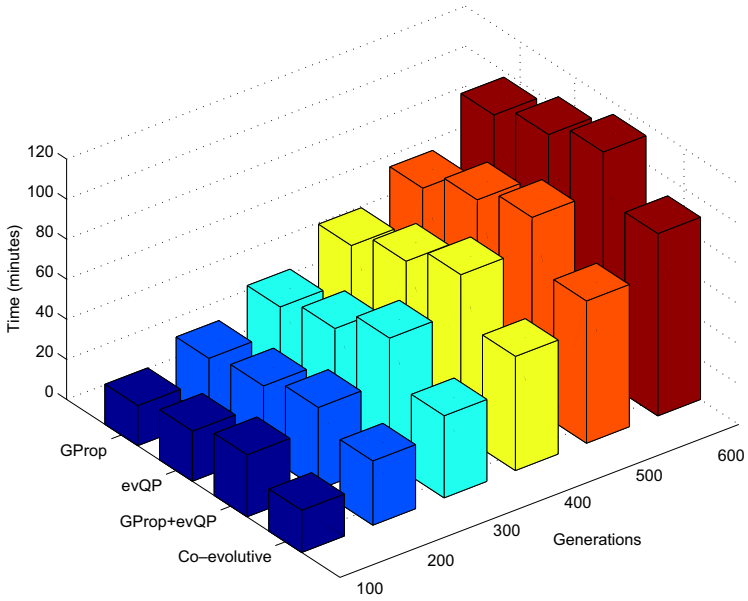
The method *GProp + evQP* gives classification errors similar to those obtained with the *G-Prop* method. Time is slightly worse, since this method first runs the *G-Prop* program and then the *evQP* one.

The *co-evolutive* method achieves a classification ability similar to that obtained using the *G-Prop* method, since the latter is also used for the MLPs optimization. However, running time is minimized (as in the case of the EA used to optimize QPs) due to the fact that the EA optimizes the number of training epochs.

As can be seen, there are small differences on error and number of training epochs between experiments using 500 and 600 generations. Generalization performance stops improving around generation 500, staying almost constant afterwards.

Results were verified using *t*-Student statistical tests. In the case of classification error, significant differences were found when the confidence level was 90% or 95%. In some cases, the means were so similar that no significant differences were found. In most cases, the resultant times presented significant differences when the confidence level was 95% (even as high as 99% in some cases). Differences were significant when the confidence level was 99% between methods, in terms of the number of training epochs obtained.

Table 3
Running time in minutes, obtained after running the experiments (Glass-I)



Generations	GProp	evQP	GProp + evQP	Co-evolutive
100	**20 ± 2***	25 ± 7	31 ± 5	21 ± 3
200	**30 ± 5***	34 ± 5	41 ± 4	32 ± 4
300	42 ± 7	49 ± 7	62 ± 6	**41 ± 5**
400	59 ± 8	69 ± 8	80 ± 5	**57 ± 6**
500	74 ± 4	86 ± 6	95 ± 6	**71 ± 6***
600	97 ± 8	105 ± 6	114 ± 6	**91 ± 4***

The best results are highlighted in order to ease interpretation (asterisks indicate significant differences). Plot shows how the running time is reduced with the number of generations. The GProp + evQP method obtains worse results, while the co-evolutive method takes less time compared to the other methods.

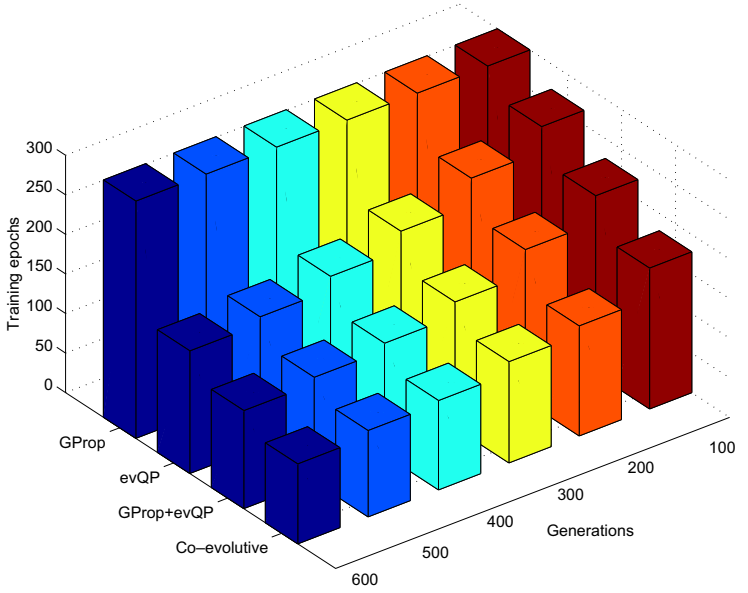
The worst results, in terms of classification ability, are obtained by applying *QP alone*, since neither architecture nor running parameters optimization are performed (see Table 5). Some authors report comparable errors on these problems [27,28], although no information on the experimental setup is given. Moreover, standard deviation is not reported, and therefore, no comparison can be made in order to apply *t*-Student tests. In any case, the best result found using the co-evolutive model, for each experimental setup, beats those presented in [27,28]. Other results found [92,46,39,8,31,63,44,42] are comparable and even worse than those presented in this paper.

5.2. Cancer-I

Results obtained for this classification problem are shown in Tables 6–8. The best results have been highlighted in order to ease interpretation (asterisks indicate significative differences with respect the other results). Tables show that, for the Cancer problem, the methods proposed behave similarly to when they were applied in the first problem (Glass).

As in the previous problem, G-Prop increases its classification ability with the number of generations. In the case of *evQP* method, classification ability is worse because no MLP optimization is carried out. *GProp + evQP* method obtained results similar to those from the *G-Prop* and *evQP* methods, although running time was slightly higher. Finally, the results obtained using the co-evolutive method, as far as classification ability is concerned, were similar to those obtained using previous methods. However, there was a reduction in running time due to the fact that this EA optimized the number of training epochs.

Table 4
Number of training epochs obtained after running the experiments (Glass-I)



Generations	GProp	evQP	GProp + evQP	Co-evolutive
100	300	268 ± 12	225 ± 15	$\mathbf{178 \pm 15}^{*}$
200	300	237 ± 16	191 ± 17	$\mathbf{139 \pm 18}^{*}$
300	300	204 ± 14	159 ± 13	$\mathbf{128 \pm 14}^{*}$
400	300	181 ± 17	141 ± 10	$\mathbf{113 \pm 12}^{*}$
500	300	164 ± 13	132 ± 12	$\mathbf{109 \pm 16}^{*}$
600	300	155 ± 16	124 ± 14	$\mathbf{101 \pm 12}^{*}$

The best results are highlighted in order to ease interpretation (asterisks indicate significant differences). Plot shows that those methods that optimize the number of training epochs reduce this parameter with the number of generations. As can be seen, GProp uses a constant value for this parameter and the co-evolutive method outperforms the other methods.

No improvement is noted on error and number of training epochs between experiments using 500 and 600 generations. It seems that generalization performance stops improving around generation 500, confirming previous results.

t-Student tests were used to verify the results. Significant differences were found between classification errors for the co-evolutive method when using 500 and 600 generations. Differences in running times were significant to levels of 90% and 95%, but in some cases, no significant differences were noted. Nevertheless, in most cases, differences in terms of number of training epochs obtained were significant when the confidence level was 99%.

Worse results, in terms of classification ability, are obtained by applying *QP alone* and even other complex methods [92,46,14,8,31,63,42] (see Table 9).

5.3. Results obtained using other partitions

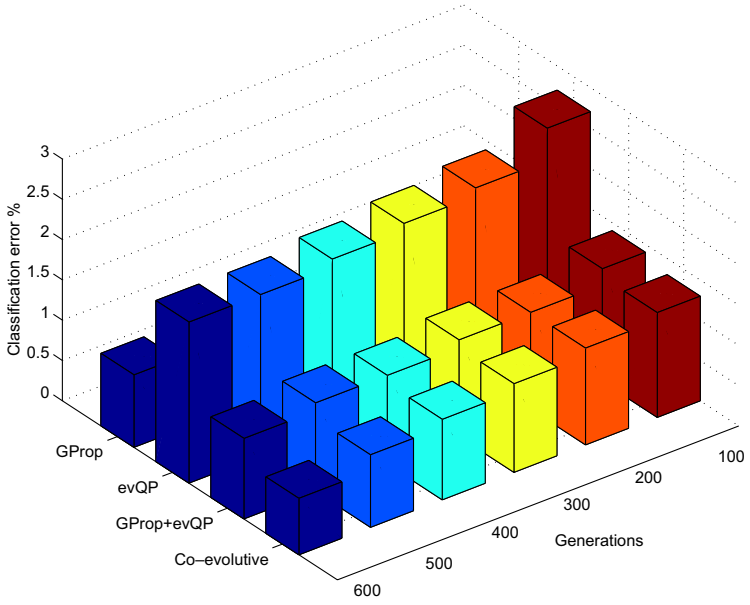
In this section, we will apply the proposed methods to the partitions II and III of the Glass and Breast Cancer datasets (proposed by Prechelt in [92]), in order to complete the experiments and verify the results obtained above. In these experiments, the number of generations was fixed to 600 (since that value yielded the best results above).

For each problem, a table is reported showing the error rate, running time, and number of training epochs using the proposed methods. Results shown in Tables 10 and 11 are obtained as the mean and standard deviation.

Table 5
Comparison of the method introduced in this paper (co-ev. method, first column) and classification ability obtained using other methods (Glass-I)

Co-ev. method	QP alone	ADA-MENN [28]	K-NN [28]	C4.5 [28]	Coop. Ens. [42]	Dzeroski and Zenko [31]	Scyth [28]	Prechelt [92]	Grönroos [46]	ARGEN + AREPO [63]	Cascade Ens. [44]	MOBNET [39]	Cantú-Paz and Kamath [8]
25 ± 3	39 ± 3	24.8	28.0	31.8	$\mathbf{22.9 \pm 4.8}$	25.2	27.1	32.08	32 ± 0.5	32.33	27 ± 3	29.6 ± 3.1	32.9

Table 6
Generalization error rate obtained after running the experiments (Cancer-I)



Generations	GProp	evQP	GProp + evQP	Co-evolutive
100	$\mathbf{1.2 \pm 0.4}$	2.7 ± 0.2	1.4 ± 0.3	1.3 ± 0.3
200	$\mathbf{1.1 \pm 0.5}$	2.3 ± 0.3	1.2 ± 0.4	1.2 ± 0.5
300	$\mathbf{1.1 \pm 0.3}$	2.2 ± 0.1	1.2 ± 0.3	$\mathbf{1.1 \pm 0.4}$
400	$\mathbf{1.0 \pm 0.3}$	2.1 ± 0.2	1.1 ± 0.3	$\mathbf{1.0 \pm 0.2}$
500	1.0 ± 0.2	2.0 ± 0.3	1.1 ± 0.2	$\mathbf{0.9 \pm 0.2}^{*}$
600	0.9 ± 0.2	2.0 ± 0.2	1.0 ± 0.3	$\mathbf{0.7 \pm 0.2}^{*}$

The best results are highlighted in order to ease interpretation (asterisks indicate significant differences regarding the other results). Plot shows how the classification error is reduced with the number of generations. The evQP method obtains worse results, while the other methods obtain comparable results.

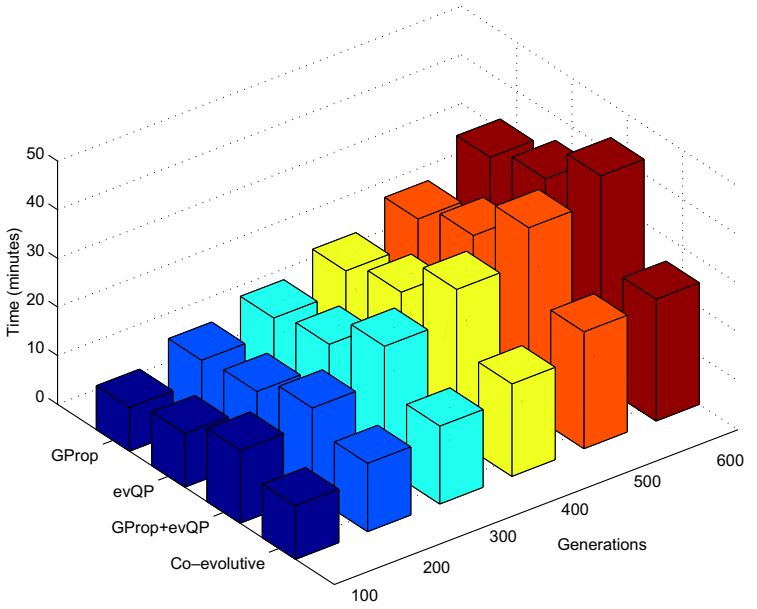
As can be seen, on the Glass-II and Glass-III problems, the co-evolutive method obtains the highest generalization ability. At the same time, the number of training epochs to calculate the MLP fitness is reduced. This results in a lower execution time, compared to the other methods.

The right-hand side of Table 10 shows the results found in the bibliography. It can be seen that the co-evolutive method has obtained comparable and even better results than those presented by other authors [92,39].

As in the problems shown in the previous sections, the co-evolutive method outperforms other methods on the Cancer-II and Cancer-III problems. It exhibits higher generalization ability and optimizes the number of training epochs. Time to complete the experiments is lower compared to the other methods.

Table 11 shows the results obtained using the QP algorithm and those published by Prechelt in [92]. As can be seen, co-evolutive method obtains comparable and even better results than those found in the bibliography.

Table 7
Running time in minutes, obtained after running the experiments (Cancer-I)



Generations	GProp	evQP	GProp + evQP	Co-evolutive
100	$\mathbf{9 \pm 2}^{*}$	11 ± 2	15 ± 4	11 ± 3
200	$\mathbf{13 \pm 1}^{*}$	14 ± 5	18 ± 5	14 ± 3
300	$\mathbf{16 \pm 3}$	18 ± 7	25 ± 2	$\mathbf{16 \pm 4}$
400	20 ± 2	23 ± 6	31 ± 5	$\mathbf{19 \pm 4}$
500	25 ± 3	29 ± 5	38 ± 4	$\mathbf{24 \pm 4}^{*}$
600	32 ± 4	35 ± 4	43 ± 4	$\mathbf{25 \pm 2}^{*}$

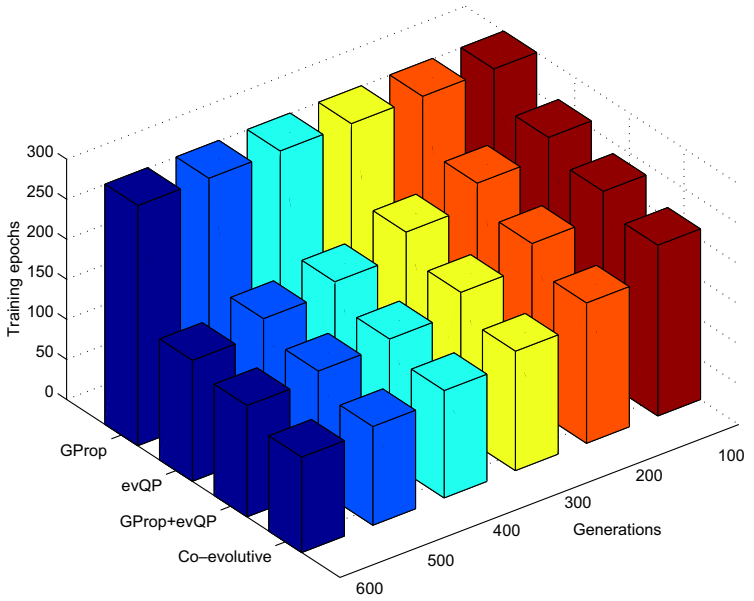
The best results are highlighted in order to ease interpretation (asterisks indicate significant differences). Plot shows how the running time is increased with the number of generations. The GProp + evQP method obtains slightly worse results, while the co-evolutive method takes less time compared to the other methods.

6. Conclusions and work in progress

In this paper, we have presented a comparison between several hybrid methods for MLP optimization. Results obtained in this paper ratify those previously published, so that we can draw the following conclusions:

- Evolutive methods obtain better results in terms of classification ability and running time than simpler methods.
- Co-evolutive method obtains lower running times than the sequential methods, and comparable or even slightly better classification ability results.
- Errors given using *GProp + evQP* are similar to those obtained using *G-Prop*, since this method optimizes the network classification ability before the training parameters. In this case, time taken is slightly higher because in the initial phase, a *G-Prop* program is used to obtain several MLPs and then, the *evQP* program is run using the networks.
- Differences between the proposed methods are significant after the application of *t*-Student statistical tests.
- Running a number of previously selected settings for the network and training parameters leads to worse results. This is because a blind search is carried out. By using an evolutive method, we can take advantage of the recombination of solutions, and therefore, no MLP parameters need to be established manually.

Table 8
Number of training epochs obtained after running the experiments (Cancer-I)



Generations	GProp	evQP	GProp + evQP	Co-evolutive
100	300	259 ± 14	236 ± 18	$\mathbf{213 \pm 12}^{*}$
200	300	236 ± 17	205 ± 13	$\mathbf{175 \pm 15}^{*}$
300	300	209 ± 12	178 ± 16	$\mathbf{149 \pm 13}^{*}$
400	300	181 ± 18	154 ± 13	$\mathbf{134 \pm 11}^{*}$
500	300	169 ± 16	148 ± 11	$\mathbf{123 \pm 12}^{*}$
600	300	151 ± 15	139 ± 12	$\mathbf{119 \pm 14}^{*}$

The best results are highlighted in order to ease interpretation (asterisks indicate significant differences). Plot shows that those methods that optimize the number of training epochs reduce this parameter with the number of generations. As can be seen, GProp uses a constant value for this parameter and the co-evolutive method outperforms the other methods.

Table 9
Comparison of the method introduced in this paper (co-ev. method, first column) and classification ability obtained using other methods (Cancer-I)

Co-ev. method	QP alone	SA- PROP [14]	Prechelt [92]	Grönroos [46]	ARGEN + AREPO [63]	Coop. Ens. [42]	Dzeroski and Zenko [31]	Cantú-Paz and Kamath [8]
$\mathbf{0.7 \pm 0.2}$	3.0 ± 0.4	1.1 ± 0.5	1.149	2.0 ± 0.6	1.86	1.23 ± 0.5	1.4	3.3

The Glass and Breast Cancer [92] pattern classification problems were used. These problems differ on the number of available patterns and on the number of patterns in each class. Results on the I–III partitions proposed by Prechelt have been presented.

After comparing the results using the co-evolutive method with those obtained using simpler methods, we can see that not only computation time is reduced, but also classification error. Thus, co-evolutive method appears to be preferable, as similar reduction in computation could not be achieved using a simpler method. Some authors report comparable classification errors on these problems [27,28], although no information on the experimental setup is given. Moreover, standard deviation is not reported, so no comparison can be made using *t*-Student tests. Other results found in the bibliography are comparable and even sensibly worse [92,46,14,63,44] than those obtained using the co-evolutive method.

Table 10
Results obtained (error, time, and number of epochs) using the II and III partitions of the Glass problem

	GProp	evQP	GProp + evQP	Co-evolutive	QP	Prechelt	MOBNET
Glass-II							
Error	34 ± 3	39 ± 5	33 ± 3	$\mathbf{32 \pm 2}^{*}$	42.9 ± 3.1	52.83	35.3 ± 2.4
Time	97 ± 6	106 ± 8	118 ± 6	$\mathbf{92 \pm 5}^{*}$	–	–	–
Epochs	300	152 ± 16	122 ± 12	$\mathbf{98 \pm 15}^{*}$	–	–	–
Glass-III							
Error	35 ± 1	40 ± 5	35 ± 3	$\mathbf{33 \pm 4}^{*}$	41.1 ± 4.2	33.96	40.6 ± 4.4
Time	94 ± 7	110 ± 5	116 ± 8	$\mathbf{91 \pm 6}^{*}$	–	–	–
Epochs	300	156 ± 13	119 ± 14	$\mathbf{101 \pm 16}^{*}$	–	–	–

These experiments were carried out using 600 generations.

Table 11
Results obtained (error, time, and number of epochs) using the II and III partitions of the Breast Cancer problem

	GProp	evQP	GProp + evQP	Co-evolutive	QP	Prechelt
Cancer-II						
Error	2.3 ± 0.2	2.6 ± 0.1	2.2 ± 0.2	$\mathbf{2.0 \pm 0.3}^{*}$	2.6 ± 0.3	5.747
Time	31 ± 4	33 ± 3	41 ± 2	$\mathbf{24 \pm 4}^{*}$	–	–
Epochs	300	154 ± 12	143 ± 14	$\mathbf{122 \pm 13}^{*}$	–	–
Cancer-III						
Error	3.3 ± 0.2	3.9 ± 0.4	3.1 ± 0.2	$\mathbf{2.9} \pm 0.5^{*}$	4.8 ± 0.4	2.299
Time	32 ± 3	36 ± 4	44 ± 3	$\mathbf{25} \pm 3^{*}$	–	–
Epochs	300	155 ± 15	140 ± 11	$\mathbf{123} \pm 12^{*}$	–	–

These experiments were carried out using 600 generations.

Validity of results was proved for each method:

- The method which optimizes both the MLP architecture and initial weights obtains good classification errors. However, running time is slightly higher than that obtained with other methods, as no training parameter optimization is carried out.
- The EA which optimizes the training algorithm parameters improves running time, but yields worse results in terms of classification ability, because network architecture and initial weights are not optimized.
- Cooperative co-evolution of QPs and MLPs, and the way the evaluation is carried out makes that the optimized parameters from evolutionary population of QP algorithms can be applied on evolved population of MLPs. As shown, results support this statement.
- Finally, as the co-evolutive method optimizes the individual-QP, the average number of training epochs is reduced. This makes the MLPs training much faster, reducing the time required. At the same time, the MLPs are optimized, which improves the classification ability.

In the near future, we would like to implement and study the performance of the distributed co-evolutionary approach, whereby, along with the QP population, there are several MLP populations evolving in parallel. Thus, the running time could be substantially reduced and the space search could be more effectively explored at the same time. Along with the implementation of this approach, it would be interesting to study the learning coefficient values through the evolution and how it affects classification ability.

It has been proved that the number of training epochs can be optimized, avoiding network overfitting and reducing the running time.

Acknowledgments

This work has been supported by CICYT TIC2003-09481-C04-01 project. The authors are grateful to anonymous referees for their constructive comments and advice about the first version of our paper.

References

[1] E. Alpaydim, GAL: Networks that grow when they learn and shrink when they forget, International Journal of Pattern Recognition and Artificial Intelligence 8 (1) (1994) 391–414.
[2] P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that construct recurrent neural networks, IEEE Transactions on Neural Networks 5 (1994) 54–64.
[3] Giuliano Armano, Michele Marchesi, Andrea Murru, A hybrid genetic-neural architecture for stock indexes forecasting, Information Sciences 170 (1) (2005) 3–33.
[4] G. Bebis, M. Georgiopoulos, T. Kasparis, Coupling weight elimination with genetic algorithms to reduce network size and preserve generalization, Neurocomputing 17 (1997) 167–194.
[5] I. Bellido, G. Fernandez, Backpropagation Growing Networks: Towards Local Minima Elimination, Lecture Notes in Computer Science, vol. 540, Springer-Verlag, 1991, pp. 130–135.
[6] C.M. Bishop, Neural Networks for Pattern Recognition, Clarendon Press, Oxford University Press Inc., New York, 1996.
[7] A. Cangelosi, D. Parisi, S. Nolfi, Cell Division and Migration in a Genotype for Neural Networks, Network: Computation in Neural Systems 5 (1994) 497–515.
[8] E. Cantú-Paz, C. Kamath, Inducing oblique decision trees with evolutionary algorithms, IEEE Transactions on Evolutionary Computation 7 (1) (2003) 54–68.
[9] P.A. Castillo, M.G. Arenas, J.G. Castellano, M. Cillero, J.J. Merelo, A. Prieto, V. Rivas, G. Romero, Function approximation with evolved multilayer perceptrons, in: Nikos E. Mastorakis (Ed.), Advances in Neural Networks and Applications, Artificial Intelligence Series, World Scientific and Engineering Society Press, 2001, ISBN 960-8052-26-2, pp. 195–200.
[10] P.A. Castillo, J. Carpio, J.J. Merelo, V. Rivas, G. Romero, A. Prieto, Evolving multilayer perceptrons, Neural Processing Letters 12 (2) (2000) 115–127.
[11] P.A. Castillo, J.G. Castellano, J.J. Merelo, A. Prieto, Diseño de Redes Neuronales Artificiales Mediante Algoritmos Evolutivos. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.14, pp. 2–32 (ISSN: 1137-3601). (c) AEPIA <http://aepia.dsic.upv.es/>, 2001.
[12] P.A. Castillo, J. González, J.J. Merelo, V. Rivas, G. Romero, A. Prieto, G-Prop-II: Global optimization of multilayer perceptrons using GAs, in: Congress on Evolutionary Computation, Washington, DC, USA, vol. III, 1999, pp. 2022–2027, ISBN 0-7803-5536-9.
[13] P.A. Castillo, J. González, J.J. Merelo, V. Rivas, G. Romero, A. Prieto, G-Prop-III: Global optimization of multilayer perceptrons using an evolutionary algorithm, in: Congress on Evolutionary Computation, Genetic and Evolutionary Computation Conference, Orlando, USA, vol. I, 1999, p. 942, ISBN 1-55860-611-4.
[14] P.A. Castillo, J. González, J.J. Merelo, V. Rivas, G. Romero, A. Prieto, SA-Prop: Optimization of Multilayer Perceptron Parameters Using Simulated Annealing, Lecture Notes in Computer Science, vol. 1606, Springer-Verlag, 1999, ISBN 3-540-66069-0, pp. 661–670.
[15] P.A. Castillo, J.J. Merelo, V. Rivas, G. Romero, A. Prieto, G-Prop: Global optimization of multilayer perceptrons using GAs, Neurocomputing 35 (1–4) (2000) 149–163.
[16] P.A. Castillo, M.G. Arenas, J.J. Castillo-Valdivieso, A. Prieto, J.J. Merelo, G. Romero, Artificial Neural Networks Design Using Evolutionary Algorithms, Advances in Soft-Computing, Springer-Verlag, 2003, ISBN 1-85233-755-9, pp. 43–52.
[17] P.A. Castillo, M.G. Arenas, J.J. Merelo, V. Rivas, G. Romero, Optimisation of Multilayer Perceptrons Using a Distributed Evolutionary Algorithm with SOAP, Lecture Notes in Computer Science, vol. 2439, Springer-Verlag, 2002, pp. 676–685.
[18] P.A. Castillo, J.J. Merelo, G. Romero, A. Prieto, I. Rojas, Statistical analysis of the parameters of a neuro-genetic algorithm, IEEE Transactions on Neural Networks, 1045-9227 13 (6) (2002) 1374–1394.
[19] A. Chandra, X. Yao, Evolutionary framework for the construction of diverse hybrid ensembles, in: Proceedings of the 13th European Symposium on Artificial Neural Networks, Brugge, Belgium, 2005, pp. 253–258.
[20] A. Chandra, X. Yao, Ensemble learning using multi-objective evolutionary algorithms, Journal of Mathematical Modelling and Algorithms 5 (4) (2006) 417–445.
[21] A. Chandra, X. Yao, Evolving hybrid ensembles of learning machines for better generalisation, Neurocomputing 69 (7–9) (2006) 686–700.
[22] Yuehui Chen, Bo Yang, Jiwen Dong, Ajith Abraham, Time-series forecasting using flexible neural tree model, Information Sciences 174 (3–4) (2005) 219–235.
[23] G. Cybenko, Approximation by superpositions of sigmoids, Mathematics of Control, Signals, and Systems 2 (1989) 303–314.
[24] C. Darwin, On the Origin of Species by Means of Natural Selection, John Murray, London, 1859.
[25] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.
[26] I. de Falco, A. Iazzetta, P. Natale, E. Tarantino, Evolutionary neural networks for nonlinear dynamics modeling, in: Parallel Problem Solving from Nature 98, Lectures Notes in Computer Science, vol. 1498, 1998, pp. 593–602.
[27] C. Domeniconi, J. Peng, D. Gunopulos, Adaptive metric nearest neighbor classification, in: Proceedings of IEEE Conference on CVPR, Hilton Head Island, SC, 2000, pp. 517–522.
[28] W. Duch, Datasets used for classification: comparison of results, 2004. <http://www.phys.uni.torun.pl/kmk/projects/datasets.html>.
[29] A.F. Duprat, T. Huynh, G. Dreyfus, Toward a principled methodology for neural network design and performance evaluation in QSAR. Application to the prediction of LogP, Journal of Chemical Information and Computer Sciences 38 (4) (1998) 586–594.
[30] P. Durr, C. Mattiussi, D. Floreano. Neuroevolution with Analog Genetic Encoding, Lecture Notes in Computer Science, vol. 4193, 2006, pp. 671–680, ISBN 978-3-540-38990-3.

[31] S. Dzeroski, B. Zenko, Is combining classifiers with stacking better than selecting the best one? Machine Learning 54 (3) (2004) 255–273.
[32] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer, 2003, ISBN 3-540-40184-9.
[33] S. Fahlman, Faster-learning variations on back-propagation: an empirical study, in: Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann, 1988, pp. 38–51.
[34] S.E. Fahlman, An empirical study of learning speed in back-propagation networks, Technical report, Carnegie Mellon University, 1988.
[35] R.A. Fisher, Theory of statistical estimation, in: Proceedings of the Cambridge Philosophical Society, vol. 22, 1925, pp. 700–725.
[36] R.A. Fisher, The comparison of samples with possibly unequal variances, Annals of Eugenics 9 (1936) 174–180.
[37] K. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (3) (1989) 183–192.
[38] A.R. Gallant, H. White, There exists a neural network that does not make avoidable mistakes, Proceedings of the IEEE International Conference on Neural Networks (San Diego), vol. 1, IEEE, New York, 1988, pp. 657–664.
[39] N. García-Pedrajas, C. Hervás-Martínez, J. Munoz-Pérez, Multiobjective cooperative coevolution of artificial neural networks, Neural Networks 15 (10) (2002) 1255–1274.
[40] N. García-Pedrajas, C. Hervás-Martínez, J. Munoz-Pérez. SYMBIONT: A cooperative coevolutionary model for evolving artificial neural networks for classification, in: Studies in Fuzziness and Soft Computing Archive, 2002, pp. 341–354, ISBN 3-7908-1455-5.
[41] N. García-Pedrajas, C. Hervás-Martínez, J. Munoz-Pérez, COVNET: A cooperative coevolutionary model for evolving artificial neural networks, IEEE Transactions on Neural Networks 14 (3) (2003) 575–596.
[42] N. García-Pedrajas, C. Hervás-Martínez, D. Ortiz, Cooperative coevolution of artificial neural network ensembles for pattern classification, IEEE Transactions on Evolutionary Computation 9 (3) (2005) 271–302.
[43] N. García-Pedrajas, D. Ortiz-Boyer, A cooperative constructive method for neural networks for pattern recognition, Pattern Recognition 40 (1) (2007) 80–98.
[44] N. García-Pedrajas, D. Ortiz-Boyer, R. del Castillo-Gomáriz, C. Hervás-Martínez, Cascade ensembles, in: J. Cabestany, A. Prieto, D.F. Sandoval (Eds.), IWANN2005, LNCS, vol. 3512, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 598–603.
[45] J.J. Grefenstette, Optimization of control parameters for genetic algorithms, IEEE Transactions on Systems, Man, and Cybernetics SMC-16 (1) (1986) 122–128.
[46] M.A. Grönroos, Evolutionary Design of Neural Networks, Master of Science Thesis in Computer Science, Dept. of Mathematical Sciences, University of Turku, 1998.
[47] F. Gruau, Genetic synthesis of boolean neural networks with a cell rewriting developmental process, in: D. Whitley, J.D. Schaffer (Eds.), Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN'92), IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 55–74.
[48] F. Gruau, Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm, PhD thesis, Ecole Normale de Lyon, France, 1994.
[49] F.C. Gruau, Cellular encoding of genetic neural networks, Technical Report, LIP-IMAG Ecole Normale Superieure de Lyon, 46 Allée d'Italie 69007 Lyon, France, 1992.
[50] J. Hallinan, P. Jackway, Co-operative evolution of a neural classifier and feature subset, Lecture Notes in Computer Science 1585 (1999) 397–404.
[51] D. Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure, in: Artificial Life II, Addison Wesley, 1992, pp. 313–323.
[52] P. Husband, F. Mill, Simulated co-evolution as the mechanism for emergent planning and scheduling, in: Belew, Booker (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1991, pp. 264–270.
[53] P. Husbands, Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation, in: T. Fogarty et al. (Eds.), Evolutionary Computing, Lecture Notes in Computer Science, vol. 865, Springer-Verlag, 1994, pp. 150–165.
[54] J. Hwang, S. You, S. Lay, I. Jou, The cascade-correlation learning: a projection pursuit learning perspective, IEEE Transactions on Neural Networks 7 (2) (1996) 278–289.
[55] I. Jagielska, C. Matthews, T. Whitfort, An investigation into the application of neural networks, fuzzy logic, genetic algorithms, and rough sets to automated knowledge acquisition problems, Neurocomputing 24 (1999) 37–54.
[56] T. Jasic, H. Poh, Analysis of Pruning in Backpropagation Networks for Artificial and Real World Mapping Problems, Lecture Notes in Computer Science, vol. 930, Springer-Verlag, 1995, pp. 239–245.
[57] R. Keesing, D.G. Stork, Evolution and learning in neural networks: the number and distribution of learning trials affect the rate of evolution, in: R. Lippmann, J.E. Moody, D.S. Touretzky (Eds.), Advances in Neural Information Processing Systems 3 [NIPS Conference, Denver, Colorado, USA, November 26–29, 1990], Morgan Kaufmann, 1991, ISBN 1-55860-184-8, pp. 804–810.
[58] D. Kim, C. Kim, Forecasting time series with genetic fuzzy predictor ensemble, IEEE Transactions on Fuzzy Systems 5 (4) (1997) 523–535.
[59] W. Kinnebrock, Accelerating the standard backpropagation method using a genetic approach, Neurocomputing 6 (1994) 583–588.
[60] A.N. Kolmogorov, On the representation of continuous functions of several variables by superpositions of continuous functions of one variable and addition, American Mathematical Society Translations 28 (1963) 55–59.
[61] P. Kuchenko, SOAP::Lite. <http://www.soaplite.com>.
[62] Z.-J. Lee, C.-Y. Lee, A hybrid search algorithm with heuristics for resource allocation problem, Information Sciences 173 (1–3) (2005) 155–167.

[63] A. León-Barranco, C.A. Reyes-García, ARGEN + AREPO: Improving the search process with artificial genetic engineering, in: J. Cabestany, A. Prieto, D.F. Sandoval (Eds.), IWANN2005, LNCS, vol. 3512, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 637–645.
[64] F.H.F. Leung, H.K. Lam, S.H. Ling, P.K.S. Tam, Tuning of the structure and parameters of a neural network using an improved genetic algorithm, IEEE Transactions on Neural Networks 14 (1) (2003) 79–88.
[65] R.P. Lippmann, An introduction to computing with neural nets, IEEE ASSP Magazine 3 (4) (1987) 4–22.
[66] Y. Liu, X. Yao, T. Higuchi, Evolutionary ensembles with negative correlation learning, IEEE Transactions on Evolutionary Computation 4 (4) (2000) 380–387.
[67] O.L. Mangasarian, R. Setiono, W.H. Wolberg, Pattern recognition via linear programming: theory and application to medical diagnosis, in: Thomas F. Coleman, Yuying Li (Eds.), Large-Scale Numerical Optimization, SIAM Publications, Philadelphia, 1990, pp. 22–30.
[68] H.A. Mayer, R. Schwaiget, R. Huber, Evolving topologies of artificial neural networks adapted to image processing tasks, in: Proceedings of the 26th International Symposium on Remote Sensing of Environment, Vancouver, BC, Canada, 1996, pp. 71–74.
[69] J.J. Merelo, Evolutionary computation in Perl, in: Münich Perl Mongers (Ed.), YAPC::Europe::2002, 2002, pp. 2–22.
[70] J.J. Merelo, M. Patón, A. Cañas, A. Prieto, F. Morán, Genetic optimization of a multilayer neural network for cluster classification tasks, Neural Network World 3 (1993) 175–186.
[71] J.J. Merelo, M. Patón, A. Cañas, A. Prieto, F. Morán, Optimization of a Competitive Learning Neural Network by Genetic Algorithms, Lecture Notes in Computer Science, vol. 686, Springer-Verlag, 1993, pp. 185–192.
[72] J.J. Merelo, A. Prieto, G-LVQ, a combination of genetic algorithms and LVQ, in: N.C. Steele, D.W. Pearson, R.F. Albrecht (Eds.), Artificial Neural Nets and Genetic Algorithms, Springer-Verlag, 1995, pp. 92–95.
[73] J.J. Merelo, P.A. Castillo, M.G. Arenas, G. Romero, Specifying Evolutionary Algorithms in XML, Lecture Notes in Computer Science, LNCS, 0302-9743 2686 (2003) 502–509.
[74] J.J. Merelo, A. Prieto, F. Morán, Optimization of classifiers using genetic algorithms, in: Mukesh J. Patel, Vasant Honavar, Karthik Balakrishnan (Eds.), Advances in the Evolutionary Synthesis of Intelligent Agents, MIT Press, 2001, ISBN 0262162016, pp. 91–108 (Chapter 4).
[75] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, third extended ed., Springer-Verlag, 1996.
[76] M. Monirul-Islam, X. Yao, K. Murase, A constructive algorithm for training cooperative neural network ensembles, IEEE Transactions on Neural Networks 14 (4) (2003) 820–834.
[77] D.J. Montana, L. Davis, Training feedforward neural networks using genetic algorithms, in: Proceedings of the 11th International Joint Conference on Artificial Intelligence, 1989, pp. 762–767.
[78] D. Moriarty, R. Miikkulainen, Hierarchical evolution of neural networks, in: Proceedings of the 1998 IEEE Conference on Evolutionary Computation (ICEC-98, Anchorage, AK), IEEE, Piscataway, NJ, 1998, pp. 428–433.
[79] D.E. Moriarty, R. Miikkulainen, Efficient reinforcement learning through symbiotic evolution, Machine Learning 22 (1996) 11–32.
[80] D.E. Moriarty, R. Miikkulainen, Forming neural networks through efficient and adaptive coevolution, Evolutionary Computation 4 (5) (1998) 373–399.
[81] B. Olsson, Co-evolutionary search in asymmetric spaces, Information Sciences 133 (3–4) (2001) 103–125.
[82] P. Palmes, S. Usui, Robustness, evolvability and optimality in evolutionary neural networks, Biosystems 82 (2) (2005) 168–188.
[83] J. Paredis, Coevolutionary computation, Artificial Life 2 (1995) 355–375.
[84] J. Paredis, The symbiotic evolution of solutions and their representations, in: Eshelman (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco, CA, 1995, pp. 359–365.
[85] R. Parekh, J. Yang, V. Honavar, MUpstart – A constructive neural network learning algorithm for multi-category pattern classification, in: Proceedings of the IEEE/INNS International Conference on Neural Networks (ICNN'97), Houston, TX, vol. III, 1997, pp. 1924–1929.
[86] R. Parekh, J. Yang, V. Honavar, Pruning strategies for the MTiling constructive learning algorithm, in: Proceedings of the IEEE/INNS International Conference on Neural Networks (ICNN'97), Houston, TX, vol. III, 1997, pp. 1960–1965.
[87] R. Parekh, J. Yang, V. Honavar, Constructive theory refinement in knowledge based neural networks, in: Proceedings of the IEEE/INNS International Joint Conference on Neural Networks (IJCNN'98), Anchorage, AK, 1998, pp. 2318–2323.
[88] R. Parekh, J. Yang, V. Honavar, Constructive neural network learning algorithms for pattern classification, IEEE Transactions on Neural Networks 11 (2) (2000) 436–451.
[89] M. Pelillo, A. Fanelli, A Method of Pruning Layered Feed-Forward Neural Networks, Lecture Notes in Computer Science, vol. 686, Springer-Verlag, 1993, pp. 278–283.
[90] V. Petridis, S. Kazarlis, A. Papaikonomu, A. Filelis, A hybrid genetic algorithm for training neural networks, Artificial Neural Networks 2 (1992) 953–956.
[91] M.A. Potter, K.A. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (1) (2000) 1–29.
[92] L. Prechelt, PROBEN1 – A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, September 1994.
[93] R.D. Reed, Pruning algorithms – a survey, IEEE Transactions on Neural Networks 4 (5) (1993) 740–744.
[94] R.D. Reed, R.J. Marks II, Neural Smithing, Bradford, The MIT Press, Cambridge, MA, London, England, 1999.
[95] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: the RPROP algorithm, IEEE International Conference on Neural Networks (San Francisco), vol. 1, IEEE, New York, 1993, pp. 586–591.
[96] C.D. Rosin, R.K. Belew, New methods for competitive coevolution, Evolutionary Computation 5 (1) (1997) 1–29.

[97] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error backpropagation, in: D.E. Rumelhart, J.L. McClelland, The PDP Research Group (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, MIT Press, Cambridge, MA, 1986, pp. 318–362.
[98] R. Smalz, M. Conrad, Combining evolution with credit apportionment: a new learning algorithm for neural nets, Neural Networks 7 (2) (1994) 341–351.
[99] D. Thierens, J. Suykens, J. Vandewalle, B. De Moor, Genetic weight optimization of a feedforward neural network controller, in: Proceedings of the Conference on Artificial Neural Nets and Genetic Algorithms, Springer-Verlag, 1993, pp. 658–663.
[100] D. Whitley, The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best? in: J.D. Schaffer (Ed.), Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kauffmann, 1989, pp. 116–121.
[101] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82.
[102] S. Yang, X. Yao, Experimental study on population-based incremental learning algorithms for dynamic optimization problems, Soft Computing 9 (11) (2005) 815–834.
[103] X. Yao, Evolving artificial neural networks, Proceedings of the IEEE 87 (9) (1999) 1423–1447.
[104] X. Yao, Y. Liu, Towards designing artificial neural networks by evolution, Applied Mathematics and Computation 91 (1) (1998) 83–90.
[105] X. Yao, Y. Xu, Recent advances in evolutionary computation, Journal of Computer Science and Technology 21 (1) (2006) 1–18.
[106] Q. Zhao, Co-evolutionary learning of neural networks, Journal of Intelligent and Fuzzy Systems, 1064-1246 6 (1998) 83–90.
[107] Q.F. Zhao, O. Hammami, K. Kuroda, K. Saito, Cooperative co-evolutionary algorithm – how to evaluate a module? in: Proceedings of the 1st IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, San Antonio, 2000, pp. 150–157.