Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 21 (2008) 1363–1379
Neural Networks
www.elsevier.com/locate/neunet

Clustering and co-evolution to construct neural network ensembles: An experimental study

Fernanda L. Minku[a,*], Teresa B. Ludermir[b]

[a] School of Computer Science, The University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK
[b] Informatics Center, Federal University of Pernambuco, Recife, 50732-970, P.O.Box 7851, Brazil

Received 12 May 2007; received in revised form 15 February 2008; accepted 20 February 2008

Abstract

This paper introduces an approach called **C**lustering and **Co**-evolution to **Co**nstruct **N**eural **N**etwork **E**nsembles (CONE). This approach creates neural network ensembles in an innovative way, by explicitly partitioning the input space through a clustering method. The clustering method allows a reduction in the number of nodes of the neural networks that compose the ensemble, thus reducing the execution time of the learning process. This is an important characteristic especially when evolutionary algorithms are used. The clustering method also ensures that different neural networks specialize in different regions of the input space, working in a divide-and-conquer way, to maintain and improve the accuracy. Besides, the clustering method facilitates the understanding of the system and makes a straightforward distributed implementation possible. The experiments performed with seven classification databases and three different co-evolutionary algorithms show that CONE considerably reduces the execution time without prejudicing (and even improving) the accuracy, even when a distributed implementation is not used.

Keywords: Neural network ensembles; Clustering; Evolutionary computation; Co-evolution

1. Introduction

Many learning problems have large amounts of data available for the learning process, such as genome and microarray analysis, geographic information analysis, intrusion detection, process control and text categorization. Ideally, it is desirable to consider all training instances simultaneously, to get the best possible estimation of class distribution. However, often it is not possible to load the whole training set into memory in one go (Inoue & Narihisa, 2005). Furthermore, the execution time of learning algorithms becomes very high when a great amount of data is used during the learning process. This problem is even aggravated when evolutionary algorithms are used.

This paper introduces an approach called Clustering and Co-evolution to Construct Neural Network Ensembles (CONE). This approach creates neural network ensembles in an innovative way, by explicitly partitioning the input space through a clustering method. The clustering method allows a reduction in the number of nodes of the neural networks that compose the ensemble, thus reducing the execution time of the learning process. Besides, it allows a straightforward distributed implementation, which makes it possible to divide the memory requirements among different machines and to further reduce the execution time. The clustering method also ensures that the neural networks that compose the ensemble specialize in different parts of the problem space and work in a divide-and-conquer manner, preserving and even improving accuracy.

CONE was used to construct Evolving Fuzzy Neural Network (EFuNN) (Kasabov, 2001b) ensembles.[1] EFuNNs are a class of Evolving Connectionist Systems (ECoSs) (Kasabov, 2003, 2007) which join the neural networks functional characteristics to the expressive power of fuzzy logic. They possess the following characteristics (Kasabov, Song, & Nishikawa, 2003):

* Corresponding author. Tel.: +44 121 4143736, fax: +44 121 414 4281.
E-mail addresses: F.L.Minku@cs.bham.ac.uk (F.L. Minku), tbl@cin.ufpe.br (T.B. Ludermir).

[1] It is important to observe that, although CONE was developed mainly to construct EFuNN ensembles, it can also be used to develop ensembles of other kinds of predictors.

0893-6080/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2008.02.001

- They facilitate evolving processes modeling task.
- They facilitate knowledge representation and extraction.
- Their learning is:
  - Lifelong: they learn from continuously incoming data in a changing environment during their entire existence.
  - On-line: they learn each example separately while the system operates. Usually, a system which operates in an on-line mode is also a systems which operates in a lifelong mode, and vice-versa.
  - Incremental: they learn new data without totally destroying the patterns learned before and without the need to make a new training on old and new data together.
  - Fast, possibly through just one pass of data propagation.
  - Local: they locally partition the problem space, allowing fast adaptation and tracing evolving processes over time.
- They can learn both as individual systems and as part of an evolutionary population of such systems.
- They have evolving structures and use constructive learning.
- They evolve in an open space, not necessarily of fixed dimensions.

Although EFuNNs and other ECoSs have some parameters that are tuned during learning, they also have parameters that do not change during the learning, but define it. The parameters that do not change during the learning can be called learning parameters. Through the use of different learning parameters, EFuNNs attain different performances and different weights are learned. The optimal learning parameter set usually depends on the incoming data and is difficult to chose manually, particularly when large databases or evolving processes are being modelled. So, approaches to automatically tune the learning parameters are important to get good results when using EFuNNs.

Several methods have been developed to optimize the learning parameters of ECoSs. Among them, Chan and Kasabov (2004), Kasabov et al. (2003), Minku and Ludermir (2005), Watts and Kasabov (2001) and Watts and Kasabov (2002) can be cited. All these methods use evolutionary algorithms, showing the importance of evolutionary algorithms when using EFuNNs. Besides using a clustering method, CONE uses a co-evolutionary algorithm that allows the optimization of the architecture of the members of the ensemble (in the case of EFuNNs, their learning parameters) at the same time as they learn their respective regions of the input space. This is very important, for different regions of the input space may require different topologies.

One of the important characteristics of ECoSs is that they facilitate knowledge extraction. So, it is important that an ensemble of ECoSs is able to preserve this feature. Clustering the problem space makes the system clearly understandable, instead of being a black box. In this way, when used to create ensembles of neural networks that already have clear rule extraction methods, it is easier to create rules to explain the system's behavior. Examples of how to extract rules from EFuNNs are presented in Kasabov (2001b). The study of rules extraction from ensembles created by CONE is proposed as a future work.

This paper presents experiments which show that CONE considerably reduces the execution time of the learning process even when CONE is not used in a distributed way. By using CONE in a distributed way, it would be possible to reduce even more the execution time. The experiments also show that the accuracy of the ensembles generated by CONE is similar or higher than the accuracy of a single EFuNN created using an evolutionary algorithm, i.e., the techniques used to reduce the execution time do not prejudice the accuracy of the system.

The objective of this paper is to introduce CONE and show that it reduces considerably the execution time to create neural networks when evolutionary algorithms are used, without prejudicing (and even being able to improve) the accuracy. In order to do that, CONE's behaviour is analysed in off-line mode, although CONE can also be applied to on-line learning if an on-line co-evolutionary algorithm is adopted. It is beyond the scope of this paper to show whether EFuNNs ensembles are better than other types of ensembles. The paper also does not intend to determine the best evolutionary algorithm to be used with CONE. Instead, it presents 3 different evolutionary algorithms to evaluate CONE's performance with different algorithms. It can be seen that CONE's reduction in execution time without prejudicing the accuracy occurs with all the co-evolutionary algorithms used in the experiments.

The paper is organized as follows: Section 2 explains some works related to the automatic ensemble construction. Section 3 presents CONE. Section 4 explains a particular type of EFuNN ensemble. Section 5 presents three different co-evolutionary algorithms which can be used with CONE. Section 6 explains the experiments which have been made on seven benchmark databases and their results. Section 7 presents the conclusions and future works. The clustering method used with CONE and the EFuNN learning algorithm are presented in the appendix.

2. Related works

The task to construct ensembles of neural networks commonly includes some manual design, such as design of individual neural networks and/or division of training data. When there are experienced human experts with sufficient knowledge of the problem to be solved, manual design and a fixed ensemble may be appropriate. An example of an ensemble of neural networks manually designed is shown in Ranawana and Palade (2005). This paper proposes a multi-classifier system based on neural networks, called MultiNNProm. The system is used to identify promoters on *Escherichia coli* bacterium's Deoxyribonucleic Acid (DNA) sequences. It is constituted by four neural networks which receive the same DNA sequence as input. Each neural network codifies the DNA sequence in a different manner. The neural network outputs are passed onto a probability builder function that assigns probabilities as to whether the presented sequence is an *E. coli* promoter or not. A result combiner is used to combine the generated probabilities to produce the final result, which indicates whether the sequence is an *E. coli* promoter or not.

In Kasabov (2001a), a multi-classifier architecture called multiEFuNN, based on Evolving Fuzzy Neural Networks (EFuNNs), is presented. MultiEFuNN is partially manually designed. The number of clusters inside the multi-modular classifier is determined by the Evolving Clustering Method (ECM) (Song & Kasabov, 2002) and is used to seed each EFuNN with the rules derived using this method. The EFuNN learning parameters are pre-determined by the user and the output is by majority vote.

Nevertheless, when there is little prior knowledge about the problem to be solved, tedious trial-and-error processes are often involved in designing ensembles of neural networks. One way to avoid such processes is to adopt nature inspired learning such as evolutionary learning. An example of the use of evolutionary learning to create ensembles of neural networks is given in Yao and Liu (1998). The authors emphasize the difference between a learning system and an optimization one. The learning system is hoped to have the best generalization, which is different from minimizing an error function. Thus, the practice of selecting the best individual of the last generation as the learned system is not the best choice. As a population contains more information than the best individual, it could be used as the final learned system, i.e., it could be used as an ensemble. This idea inspired various subsequent works, which use the whole final population (Chen & Yao, 2007; Duell, Fermin, & Yao, 2006; Liu & Yao, 1998; Liu, Yao, & Higuchi, 2000) or part of the final population (Chandra & Yao, 2006; Chen & Yao, 2006; Duell et al., 2006; Liu & Yao, 1998; Liu et al., 2000) as the ensemble.

An approach called Evolutionary Ensemble with Negative Correlation Learning (EENCL) is proposed by Liu and Yao (1998) and Liu et al. (2000). The learning occurs in two levels: local learning of individual neural networks based on negative correlation learning and evolutionary learning based on Evolutionary Programming (Eiben & Smith, 2003). Negative correlation learning introduces a penalty term into the error function of individual neural networks, so that they can be trained cooperatively. Using the penalty term, the error of an individual neural network is negatively correlated to the errors of the other neural networks of the ensemble. The ensemble can be constituted by either all the individual neural networks in the last generation or selecting one representative from each species in the last generation. Differently from CONE, the species are determined by clustering the individuals of the last generation by using the k-means algorithm (MacQueen, 1967) after the evolutionary process is finished. CONE uses a clustering algorithm to divide the input space of the problem to form species **before** the evolutionary process starts, reducing the execution time of the learning process and directly ensuring that each species will specialize in different regions of the input space.

Another approach based on clustering is Li, Huang, Ye, and Cui (2004). Similarly to Liu and Yao (1998) and Liu et al. (2000), this approach also clusters the neural networks only after the training is done, instead of clustering the input space of the problem.

A constructive algorithm for training cooperative neural network ensembles (CNNEs) is presented in Islam, Yao, and Murase (2003). It emphasizes the accuracy and diversity among individual neural networks in an ensemble by using negative correlation learning and allowing different neural networks to be trained using different numbers of epochs. The number of hidden nodes in individual neural networks and the number of neural networks in the ensemble is determined in a constructive way. Thus, the approach is able to automatically determine the number of nodes in the individual neural networks and the size of the ensemble. This approach does not ensure that different neural networks will specialize in different regions of the input space, as does CONE.

A multi-objective evolutionary algorithm to construct ensembles of neural networks, called Diverse and Accurate Ensemble Learning Algorithm (DIVACE), is described in Chandra and Yao (2006). This method aims to find an optimal trade-off between diversity and accuracy by using these objectives explicitly as multi-evolutionary pressures. A multi-objective evolutionary approach yields to a set of near optimal solutions instead of just a single solution. Thus, these near optimal solutions (Pareto set) can be used as members of the ensemble. In the same way as in Islam et al. (2003), this approach does not ensure that different neural networks will specialize in different regions of the input space. Besides, the execution time of the evolutionary algorithm is not reduced, as in CONE.

An important framework based on cooperative co-evolution to evolve neural network ensembles is described in Garcia-Pedrajas, Hervas-Martinez, and Ortiz-Boyer (2005). It maintains two populations: population of networks and population of ensembles. The population of networks consists of a fixed pre-determined number of independent subpopulations of networks that are evolved using evolutionary programming. Each member of the population of ensembles is formed by a network from every network subpopulation. One of the problems of this approach is the use of a pre-determined ensemble size, instead of automatically determining the best number of ensembles to the problem.

Interesting parallels can be traced between Mixture-of-Experts (Jacobs & Jordan, 1991; Jordan & Jacobs, 1992, 1994) and CONE. Mixture-of-expert models also are composed by modules that are experts in different regions of the problem space. However, the architecture of each module is not evolved, the number of ensemble members is pre-determined and the division of the problem space is not so direct and intuitive as in CONE. An example of recent work derived from mixture-of-experts is the work done by Liao, Li, and Carin (2007), which presents a statistical learning model formulated to handle incomplete data.

CONE determines the number of neural networks which compose the ensemble by clustering the input space of the problem, similarly to Kasabov (2001a). However, CONE uses co-evolutionary learning, allowing the optimization of the parameters of the individual neural networks. In this way, an automatic design of the ensemble of neural network is performed. Section 3 presents CONE.

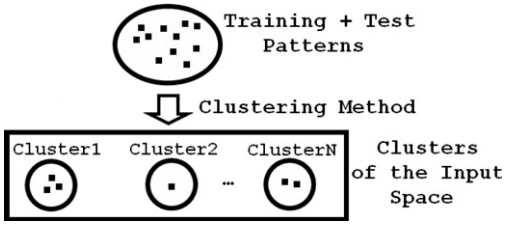

Fig. 1. Clustering.

3. CONE

This section presents CONE. The main idea of CONE is to construct ensembles of neural networks by using a clustering method to partition the input space in clusters. The clusters are used to separate the training and test patterns into various subsets of training and test patterns with empty intersection. Each subset is used to train/test a different species of individuals. The species are composed by neural networks and are evolved through a co-evolutionary algorithm. As in nature, species are genetically isolated, i.e., the individuals can only match with other individuals of the same species. Thus, each cluster is associated to a training subset, a test subset and a species of neural networks. The purpose of training the individuals of each species using a different subset of the training patterns is to specialize different species in different regions of the problem space and create diversity. Besides, as the neural networks of each species learn a reduced training set, they can be smaller than a single neural network used to learn the whole training set. In this way, each training example is presented to a relatively small neural network, reducing the training time.

Fig. 1 illustrates the creation of clusters of the input space using the training and test patterns set. It is important to notice that, although the clusters of the input space are used to create training and test subsets with empty intersection, the clusters of the input space themselves do not need to have empty intersection. Thus, a pattern could belong to more than one cluster of the input space, but it could not belong to more than one training or testing subsets.

The patterns used by the approach are divided in 3 types:

- Training patterns: used to create clusters and to train the neural networks;
- Test patterns: used to create clusters and to test the neural networks during the evolutionary process;
- Final test patterns: used to test the ensemble of neural networks generated at the end of the evolutionary process.

The training + test data set is subdivided into subsets according to the clusters of the input space, as Fig. 2 shows. In this way, if there are N clusters, there will be N subsets of training patterns with empty intersection and N subsets of test patterns with empty intersection. If there are not enough patterns to compose a test subset corresponding to a specific cluster of the input space, the same subset used to train the neural networks of the corresponding species can be used to test them during the evolutionary process.

If there are N clusters, there will be also N species to be evolved through a co-evolutionary algorithm. The individuals of the species are neural networks and the co-evolutionary algorithm can be used to optimize their parameters. These parameters can be both the architecture of the neural networks and, for example, the weights of the connections. Thus, it is possible to use the co-evolutionary algorithm both to train and to optimize the architecture of the neural networks. However, it is also possible to use the specific learning algorithm of the neural networks to train them and the co-evolutionary algorithm just to optimize their architectures, as it is done to create EFuNN ensembles (Section 4). It is important to emphasize that each training/test subset is used to train/test the individuals of a specific species.

The evolutionary process is cooperative because the fitness of an individual of a species population is calculated using a representative individual of each one of the other species populations to constitute an ensemble of neural networks. The representative of a population could be, for example, its best individual. There is no matching between individuals of different species. The interaction among individuals of different species only occurs in the calculation of the fitness value.

At the end of the evolutionary process, the representatives of each species population of the last generation are used to constitute the ensemble, as shown by Fig. 2. Fig. 3 illustrates an ensemble of neural networks created after the evolutionary process and the way it is used/tested. In order to use/test the ensemble, the clusters to which the input test pattern belongs are determined. After that, the outputs of the EFuNNs corresponding to these clusters are calculated and combined using a pre-determined combining method. Examples of combining methods can be found in Dietterich (1998).

When an on-line clustering method and an on-line co-evolutionary algorithm are used, it is possible to apply CONE to perform on-line learning. On-line co-evolutionary algorithms can be created based on algorithms such as those proposed by Chan and Kasabov (2004) and Minku and Ludermir (2005). In on-line learning mode, the system learns data which arrive in batches and the species are created when new clusters are generated. After the learning of each new batch of data, the representatives of each population are used to compose the ensemble. When a new batch of data is received, the neural networks that compose the existent species can be initialized with the weights learnt by the representatives generated through the previous batch of data.

3.1. A distributed implementation

CONE allows a straightforward distributed implementation. In this way, the memory requirements can be divided among different machines and the execution time can be reduced even more than when CONE is not executed in a distributed way. After the clustering of the training + test patterns set is made, each subset of patterns can be sent to a different machine, which evolves a different species. Communication among the machines is necessary only to calculate the fitness

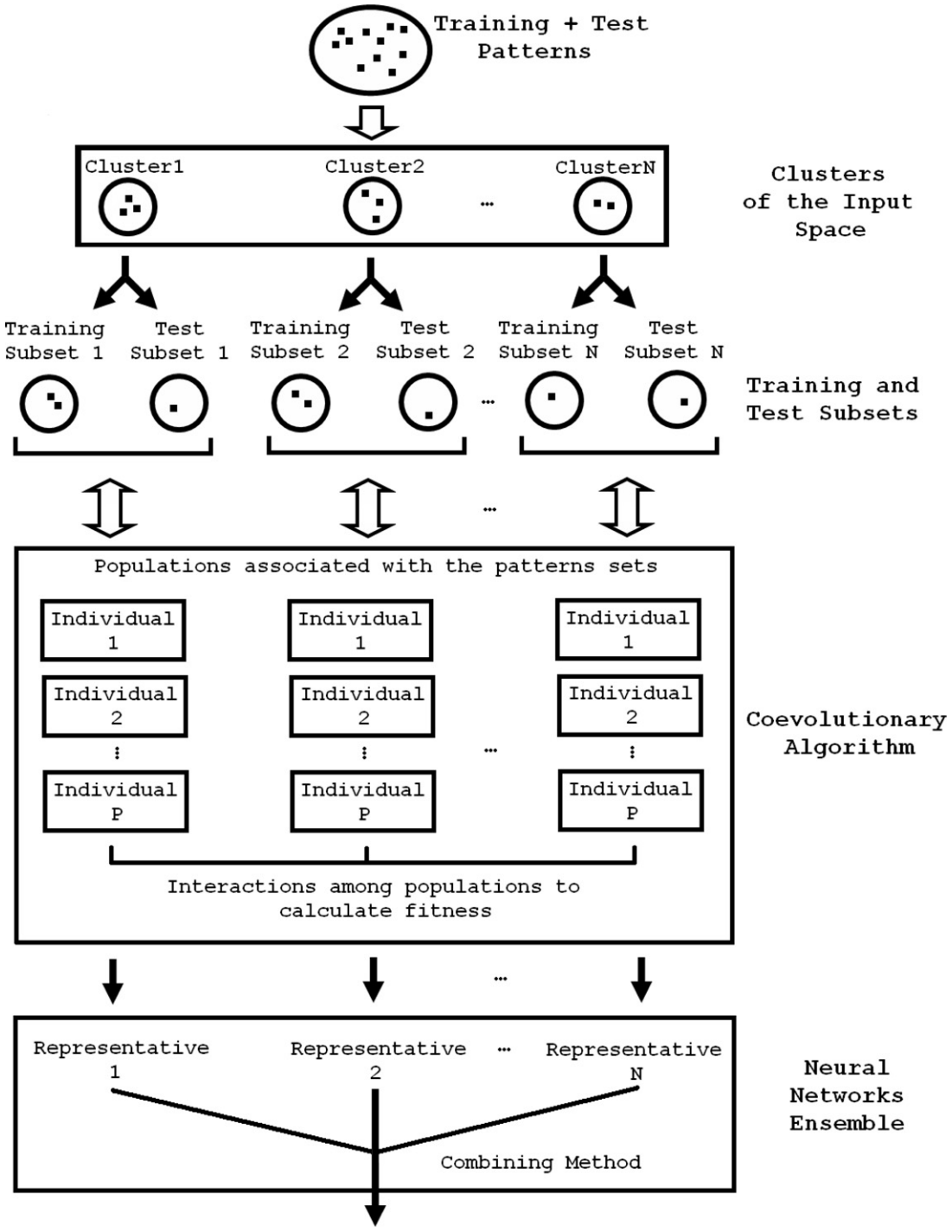

Fig. 2. CONE.

of the individuals. Besides, only values corresponding to the representative of the last generation which finished being executed have to be sent from one species to the others (e.g., error and size of the representative), not overloading the network traffic. Each machine starts executing a new generation only after it received the representatives' information of all the other species. In this way, it is ensured that it is possible to calculate the fitness values when a new generation starts. The execution time is lower bounded by the sum of the highest execution time of each generation and the communication time. The execution of CONE in a distributed way is proposed as a future work.

4. An example of EFuNN ensemble

This section presents an example of EFuNN ensemble which can be created by CONE. As it was explained in Section 1, EFuNNs have parameters which are adjusted during the learning and learning parameters. In order to create EFuNN ensembles using CONE, the co-evolutionary algorithm can

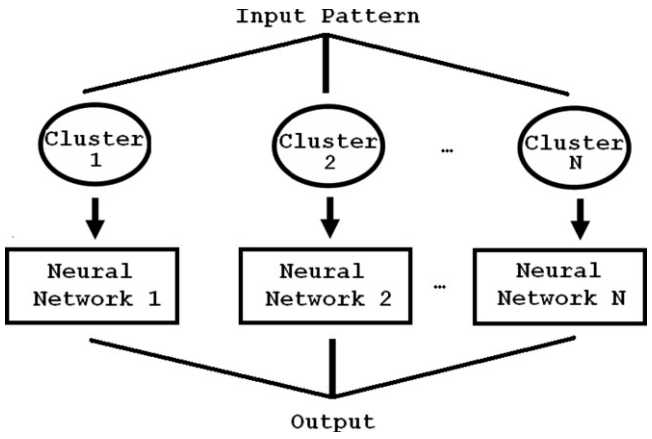

Fig. 3. Neural network ensemble utilization.

be used to optimise the learning parameters and the EFuNN learning algorithm itself can be used to train the EFuNNs.

According to CONE, the representatives of the last population of each species are used to construct the EFuNN ensemble after the evolutionary process. The best fit individual of a population can be considered the representative of this population. Examples of combining methods which can be used to combine the outputs of the EFuNNs which compose the ensemble are:

- Arithmetic average of the outputs of the EFuNNs corresponding to the cluster to which the presented pattern belongs.
- Weighted average of the outputs of the EFuNNs corresponding to the cluster to which the presented pattern belongs. The value to be used as the weight of the cluster C_j, $j = 0, 1, \ldots, N$ is $1/\|x_i - Cc_j\|$, where x_i is the presented pattern and Cc_j is the cluster center.

If a pattern does not belong to any cluster, the output of the ensemble is the output of the EFuNN corresponding to the cluster whose center is the nearest center to the pattern.

5. Co-evolutionary algorithms

This section presents 3 different co-evolutionary algorithms that can be used with CONE: a co-evolutionary genetic algorithm, a multi-objective co-evolutionary genetic algorithm and a multi-objective co-evolutionary strategy.

5.1. Co-evolutionary Genetic Algorithm

This section presents a co-evolutionary genetic algorithm (co-evolutionary GA) inspired on Potter and DeJong (2000) to be used with CONE. The co-evolutionary GA has a binary representation of the EFuNN parameters to be optimized, bitwise bit-flipping mutation, one-point crossover, and generational survivor selection. The parents selection is proportional to the value determined by be following equation:

$$Prob_{i,p,g} = \frac{max\_fitness_{p,g} - fitness_{i,p,g}}{\sum_{j=0}^{pop\_size_p - 1} fitness_{j,p,g}}, \quad (1)$$

where $max\_fitness_{p,g}$ is the greatest fitness of the population p of the generation g, $fitness_{i,p,g}$ is the fitness of the individual i of the population p of the generation g, and pop_size_p is the size of the population p.

The parent selection uses the roulette wheel method and, according to Eq. (1), the fitness value is minimized. Eiben and Smith (2003), a recent book about evolutionary algorithms, gives examples of applications of GAs in which the fitness function is directly minimized, instead of changing it into a function that has to be maximized.

Each initial population is composed by individuals created randomly choosing values for each of the EFuNN parameters to be optimized. A population is created for each cluster of the input space, i.e., for each species, according to CONE.

In the initial population, the fitness of the individuals is calculated in an isolated manner, i.e., to determine the fitness of an individual, the individuals of the other species are not considered. The function used to calculate the fitness in this generation is:

$$fitness_i = W_{rmse}\, RMSE_i + W_{size}\, size_i, \quad (2)$$

where W_{rmse} and W_{size} are pre-defined weights, $RMSE_i$ is the Root Mean Squared Error (RMSE) obtained testing the EFuNN corresponding to the individual i with the test subset corresponding to its species, and $size_i$ is the size of this EFuNN.

The size component of the fitness function is used to penalize the size of the EFuNNs generated, as suggested by Minku and Ludermir (2005). In this way, the execution time of the evolutionary algorithm is not so high.

In all generations after the initial one, the fitness of an individual i is calculated in an innovative way. Instead of playing the individual against the individuals of the other species to form an ensemble and using the individual's contribution to the ensemble as a fitness measure (as it is usually done in the literature), the fitness of an individual is the combination of the output error and the size of this individual with the output error and size of the representatives of the other species' populations of the previous generation, saving time during the fitness calculation. The functions used to calculate the fitness in all generations except the initial generation are:

$$RMSE = \sqrt{\frac{SSE_i + repr\_sse}{total\_test\_patterns\_number}} \quad (3)$$

$$size = size_i + repr\_size \quad (4)$$

$$fitness_i = W_{rmse}\, RMSE + W_{size}\, size, \quad (5)$$

where W_{rmse} and W_{size} are pre-defined weights, SSE_i is the Sum of Squared Error (SSE) obtained testing the EFuNN corresponding to the individual i with the test subset corresponding to its species, $size_i$ is the size of this EFuNN, $repr\_sse$ is the sum of the SSEs and $repr\_size$ is the sum of the sizes of the representatives of all other species' populations in the previous generation, and $total\_test\_patterns\_number$ is the total number of test patterns, including the patterns corresponding to all species.

Each species is evolved in a separate manner and there is interaction among the species only to calculate the fitness value,

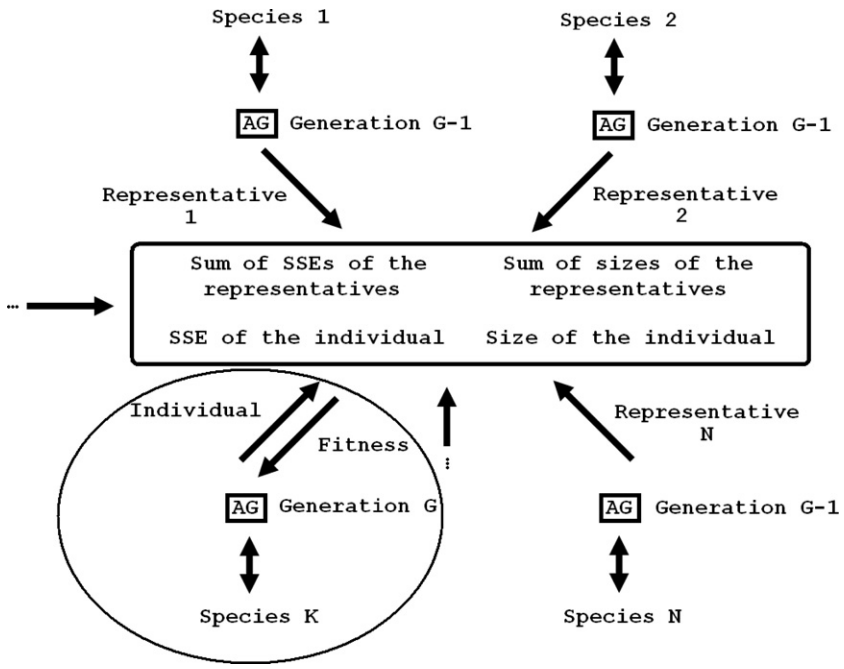

Fig. 4. Fitness calculation of an individual of the population of the species K, in a co-evolutionary GA with N species.

according to CONE. The fitness value of an individual of a species' population depends on the representative individuals of the other species' populations. Fig. 4 shows the calculation of the fitness of an individual of the population of species K of generation G, in a co-evolutionary GA with N species.

The Algorithm 5.1 is the algorithm used to evolve a specific species. The stop criterion used for the evolutionary process is the number of generations.

Algorithm 5.1 (*Evolutionary process of a species*).

(i) Create the initial population.
(ii) Repeat until a maximum number of generations is attained:
  (a) If the EFuNNs corresponding to the individuals of the population have any rule nodes, delete them.
  (b) Apply the EFuNN learning algorithm to each EFuNN of the population using the training subset corresponding to the species and the parameters codified by the genotype of the individual.
  (c) Test the EFuNNs corresponding to all individuals of the population using the test subset.
  (d) Determine the fitness value of each individual of the population.
  (e) Make parent selection using a roulette wheel method and probabilities determined through the Eq. (1).
  (f) Apply crossover and mutation with probabilities *Pc* and *Pm*, respectively, to generate new individuals.
  (g) Apply generational survivor selection.

5.2. Multi-objective co-evolutionary Genetic Algorithm

The algorithm described in the Section 5.1 needs the predefinition of the fitness parameters W_{rmse} and W_{size}. The choice of the best parameters to be used is difficult and influences the ensemble of neural networks generated after the evolutionary process. So, this section presents a multi-objective co-evolutionary GA to be used with CONE. The difference between this algorithm and the one described in Section 5.1 is that the parent selection is based on the rank of the individual in its population, instead of being based on the fitness value.

The rank of each individual is based on the objective functions, which are similar to the components of the fitness functions described in Section 5.1. The objectives vector of an individual i of the initial population is:

$$[RMSE\_Obj_i = RMSE_i,\ SIZE\_Obj_i = size_i]. \tag{6}$$

The objectives vector of an individual i in all other generations is $[RMSE\_Obj_i,\ SIZE\_Obj_i]$, where:

$$RMSE\_Obj_i = \sqrt{\frac{SSE_i + repr\_sse}{total\_test\_patterns\_number}} \quad \text{and} \tag{7}$$

$$SIZE\_Obj_i = size_i + repr\_size. \tag{8}$$

The rank of an individual i of the population p in the generation g is the number of individuals $j \neq i,\ j \in p$ which dominate the individual i in the generation g, as it is done in Fonseca and Fleming (1998). An individual i dominates an individual j if ($RMSE_i \leq RMSE_j$) and ($SIZE_i \leq SIZE_j$) and ($RMSE_i < RMSE_j$ or $SIZE_i < SIZE_j$). In this way, a Pareto optimal individual (an individual which is not dominated by any other individual of the population) has always rank equal to 0.

The best individual of a population is the individual with the lowest rank. When more than one individual has the same rank, the best between them is the one which has the lowest SSE, obtained testing the EFuNN corresponding to it with the test subset corresponding to its species.

5.3. Multi-objective co-evolutionary strategy

This section explains a multi-objective co-evolutionary strategy to be used with CONE.

Besides the features of a co-evolutionary algorithm to be used with CONE, the multi-objective co-evolutionary strategy has the following features:

- Representation by real values, with two genes for each EFuNN parameter to be optimized. The first of these genes represents the parameter itself and the second represents the corresponding self-adapting standard deviation.
- Gaussian perturbation mutation with self-adaptation of the standard deviations and one standard deviation for each variable representing an EFuNN parameter to be optimized. This kind of mutation is adequate for ordinal numeric parameters optimization.
- Local discrete crossover for the variables representing the parameters to be adjusted and local intermediary crossover for the variables representing the self-adapting standard deviations, as suggested in Eiben and Smith (2003).
- Random parent selection with equal probabilities for all population members.

The algorithm works in a similar way to the algorithm presented in Section 5.2, but in a evolutionary strategy way. So, the initial population of each species is composed by individuals created randomly choosing values for each of the EFuNN parameters to be optimized. N parents are randomly chosen to generate the individuals of the next population. N must be chosen in such a way that the number of children individuals is greater than the population size. After parent selection, the crossover operator is applied with probability Pc to each pair of parents. The result of the application of the crossover operation to a pair of parents is only one child individual. After crossover, Gaussian perturbation mutation is applied to the resultant individual, using the standard deviation corresponding to each EFuNN parameter. If the crossover operator was not applied to a pair of parents, mutation is applied to the first of the parents, resulting in a child individual.

The survival selection is similar to (μ, λ)-survival selection. The μ best rank individuals are selected among the λ children. μ is also the size of the population. The number of children is $\lambda = N/2$ and $\lambda > \mu$. This kind of survival selection was chosen because it is better than $(\mu + \lambda)$ in following moving optimal points on the search space, to escape from local optima and to be used with self-adaptation of the mutation parameters (Eiben & Smith, 2003).

The rank of an individual is determined in a very similar way to Section 5.2. However, it is calculated using the children individuals, before the survival selection. Thus, the rank of a child i is the number of children $j \neq i$ which dominate i. In this way, a Pareto-optimal individual (an individual that is not dominated by any other individual) is also always rank 0.

6. Experiments

This section presents the experiments and analyses made with CONE to create EFuNN ensembles (Section 4) using the algorithms presented in Section 5 and Appendix A. Seven different classification databases from UCI Machine Learning Repository (Newman, Hettich, Blake, & Merz, 1998) and Proben1 (Prechelt, 1994) were used: Cancer, Glass, Iris, Wine, Heart, Pima and Vehicle. Heart is considered a difficult database, for it has many missing values. Pima and Vehicle are considered some of the most difficult databases of the repository. Some preliminary experiments were made with Iris, Wine, Glass, Cancer (Minku & Ludermir, 2006a, 2006b). The preliminary experiments used a reduced number of executions and only co-evolutionary GA and multi-objective co-evolutionary GA.

As the co-evolutionary GA has the problem of being too much influenced by the tuning of the fitness function, this paper presents experiments only with Iris, Wine, Glass and Cancer using this co-evolutionary algorithm.

The results obtained with CONE were compared with evolutionary algorithms to generate and optimize EFuNNs. Evolutionary algorithms are techniques suggested by Kasabov (2003) to optimize EFuNNs parameters and since then they have inspired the work done by Chan and Kasabov (2004), Kasabov et al. (2003) and Minku and Ludermir (2005).

The rest of this section is organized as follows: Section 6.1 presents the objectives of the experiments, Section 6.2 shows the experimental setup, Section 6.3 presents an analysis of the execution time and the relation between the nodes number and the execution time, Section 6.4 presents an analysis of the accuracy and Section 6.5 presents a comparison between the use of multi-objective co-evolutionary GA and the multi-objective co-evolutionary strategy with CONE. The analyses show that it is possible to use CONE to reduce considerably the execution time without prejudicing (and even improving) the accuracy. This behaviour occurs even when different co-evolutionary algorithms are used. However, it is important to notice that different co-evolutionary algorithms should be used to different purposes, as it is shown in Section 6.5.

6.1. Objectives

The main objective of the experiments is to show that CONE reduces considerably the execution time of the learning process even when it is not used in a distributed way, for the clustering method makes the ensemble members have a reduced number of nodes. The execution time reduction is attained without prejudicing the accuracy and even improving it in comparison to single EFuNNs generated using evolutionary algorithms corresponding to the co-evolutionary algorithm used by CONE. It is important to notice that the single EFuNNs are strong classifiers with optimized learning parameters.

The experiments also aim at determining which combination method (weighted or arithmetic) is better to be used by an ensemble generated by CONE and at analysing the influence of the choice of the co-evolutionary algorithm.

With these objectives in mind, EFuNN ensembles generated using CONE with each of the presented co-evolutionary methods were compared with single EFuNNs generated using corresponding evolutionary algorithms. The characteristics

compared were the execution times and the classification errors. The classification errors were those obtained using the final_test patterns set to test the single EFuNNs and the EFuNN ensembles generated after the evolutionary processes. CONE using multi-objective co-evolutionary GA was compared with CONE using multi-objective co-evolutionary strategy to analyse the influence of the co-evolutionary algorithm in the learning process.

6.2. Experimental setup

The training, test and final_test data sets utilized by the experiments were created as follows:

(i) Two data sets (training + test data set and final_test data set) were created:
  – Proben1 databases (Glass and Cancer) are already separated in training (50% of the patterns), validation (25% of the patterns) and test data sets (25% of the patterns). There are also 3 different partitions of the patterns which compose each of these sets. For each partition, the supplied training and validation sets were used to compose CONE's training + test data sets and the supplied test set was used to compose CONE's final_test data set.
  – The other databases were obtained directly from the UCI Machine Learning Repository. These databases were processed in order to create 3 different partitions of training + test and final_test data sets. Each training + test data set contains 75% of the patterns of each class and each final_test data set contains 25% of the patterns of each class.

(ii) The 3 partitions of the training + test and final_test data sets were used in different executions. For each execution:
  (a) The training + test patterns set was used to create the clusters of the input space.
  (b) After that, the training + test patterns set was separated according to the clusters into subsets of training + test patterns. Each pattern belongs to the subset corresponding to the cluster which has the nearest center to this pattern.
  (c) Each training + test subset was then divided into 2 subsets. One of them is a training subset, with 66% of its patterns, and the other one is a test subset, with 34% of its patterns. In this way, the total number of training patterns is always 50%, the total number of test patterns is 25% and the total number of final test patterns is 25% of the patterns of the database.

The EFuNN parameters which were optimized during the co-evolutionary process and their intervals of allowed values are: m-of-n ($[1, 15] \in Z$), error threshold ($[0.01, 0.6] \in R$), maximum radius ($[0.01, 0.8] \in R$), initial sensitivity threshold ($[0.4, 0.99] \in R$) and membership functions number ($[2, 8] \in Z$).

The *Dthrs* parameter of the clustering method was empirically determined and it is 0.37 for Cancer, 0.20 for Glass, 0.40 for Iris, 50 for Wine, 45 for Heart, 0.26 for Vehicle and 50 for Pima. Table 1 shows the number of clusters created using the clustering method presented in Appendix A, for each of the partitions.

Table 1
Number of clusters

	Cancer	Glass	Iris	Wine	Heart	Vehicle	Pima
Partition 1	7	13	13	7	6	4	4
Partition 2	7	17	15	6	5	4	5
Partition 3	6	15	13	7	5	4	5
Average	6.6667	15	13.6667	6.6667	5.3333	4	4.6667

The parameters of the co-evolutionary GA are: population size = 12, $Pm = 2\%, Pc = 70\%$, $W_{rmse} = 0.1$, and stop criterion = 50 generations. Four different values for W_{size} were used: 0.005, 0.0005, 0.00005 and 0.000005.

The parameters of the multi-objective co-evolutionary GA are: population size = 12, $Pm = 2\%, Pc = 70\%$, and stop criterion = 50 generations.

The parameters of the multi-objective co-evolutionary strategy are: population size = 12, $Pc = 70\%$, number of children = 48, and stop criterion = 50 generations.

The genotypes of the co-evolutionary GA and of the multi-objective co-evolutionary GA were composed by 4 bits for m-of-n, 9 for error threshold, 10 for maximum radius, 6 for initial sensitivity threshold and 3 for the membership functions number of each input/output attribute.

The genotypes of the multi-objective co-evolutionary strategy were composed by 8 real type variables to represent *m-of-n*, *E*, *Mrad* and *S* and their corresponding self-adapting standard deviations, plus 2 real type variables to represent each of the number of membership functions and their corresponding self-adapting parameters.

Hence, for the co-evolutionary GA, CONE was executed with 4 different combinations of evolutionary parameters for each database. For the multi-objective algorithms, CONE was executed with 1 combination of evolutionary parameters. Three different partitions of the training + test and final_test data sets were also used, thus totaling 12 combinations of configurations for the co-evolutionary GA, and 3 combinations of configurations for each of the multi-objective algorithms. Thirty executions with different random seeds were performed for each combination, totaling 360 executions per database for the co-evolutionary GA and 90 executions per database for each of the multi-objective algorithms.

Executions with the above combinations of parameters were also made using a genetic algorithm (GA), a multi-objective GA and a multi-objective evolutionary strategy to generate a single EFuNN. These algorithms were the same as the co-evolutionary algorithms presented in Sections 5.1–5.3, respectively, but using the initial population fitness/objective functions for all generations and only one species. In this way, 360 GA executions were made for each database and 90 executions of each of the multi-objective algorithms were made for each database.

The above parameters were determined empirically and they aim at not having a too high execution time, as the objective of

Table 2
Measures related to the execution times (in seconds) for multi-objective co-evolutionary strategy

		Cancer	Glass	Iris	Wine	Heart	Pima	Vehicle
Ens	Av	345.4333	66.9333	10.6	29.1667	4018.2556	1628.5222	3401.9556
	SD	37.0105	1.7340	0.7313	1.0626	19.1073	26.3477	103.6192
	Min	287	65	3	27	3674	1276	1995
	Max	391	81	9	31	4443	1919	4167
Sing	Av	836.6111	109.1667	37.8667	95.4111	8406.4444	3465.2444	8343.3444
	SD	72.6796	1.3841	2.3900	1.9308	27.8621	3.3603	6.1166
	Min	748	106	34	92	8005	3398	8209
	Max	985	112	43	99	8917	3538.0000	8481

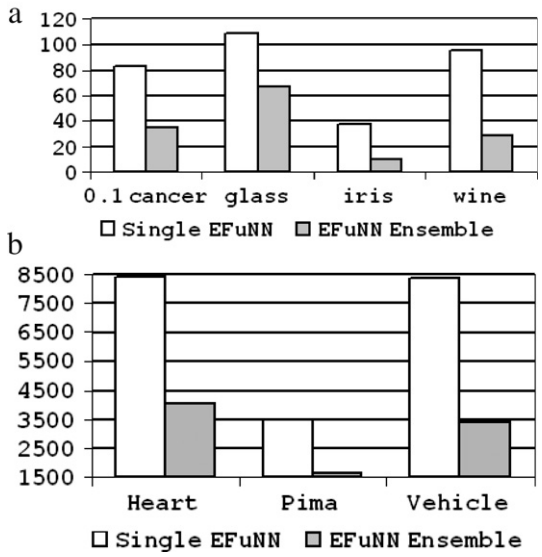

Fig. 5. Execution time average (in seconds) per database for multi-objective co-evolutionary strategy.

the paper is not finding the best co-evolutionary algorithm to be used with CONE.

6.3. Execution time

The experiments show that the execution time of CONE to generate EFuNN ensembles is always considerably lower than the execution time of the corresponding evolutionary algorithms to generate single EFuNNs.

Fig. 5 shows the execution time averages for CONE with the co-evolutionary strategy and the corresponding evolutionary strategy. Cancer execution time average is multiplied by 0.1 in this figure. Table 2 shows the execution time averages, standard deviations, minimal and maximum values.

For all databases, the execution time average of CONE to generate EFuNN ensembles was statistically lower than the execution time average of the multi-objective evolutionary strategy to generate a single EFuNN. Paired T student statistic tests (Witten & Frank, 2000) were performed to prove this analysis. For a level of significance of 0.05, the p-values of the statistic test that are less than 0.05 indicate that there is difference between the compared algorithms. The lower the p-value, the higher the confidence of the difference. This is valid for all T Student tests presented in this paper. All the p-values of the tests related to the execution time averages were 0.0000, indicating that there is significant difference between the compared algorithms. Notice that these p-values are very small.

Fig. 6 shows the division of the evolutionary algorithm's execution time by CONE's execution time. The higher this value, the higher the reduction in the execution time when CONE is used. It is possible to observe that, for all databases but Glass, the execution time of CONE was at least 2 times faster than the corresponding multi-objective evolutionary strategy, which can be considered a high reduction in the execution time.

The execution time of CONE with the other co-evolutionary algorithms was also always considerably lower than their corresponding evolutionary algorithms. Figs. 7 and 8 show the execution time averages for the other co-evolutionary algorithms. Tables 3 and 4 show the execution time averages, standard deviations, minimal and maximum values for these algorithms. The p-values of the T Student tests performed were all 0.0000.

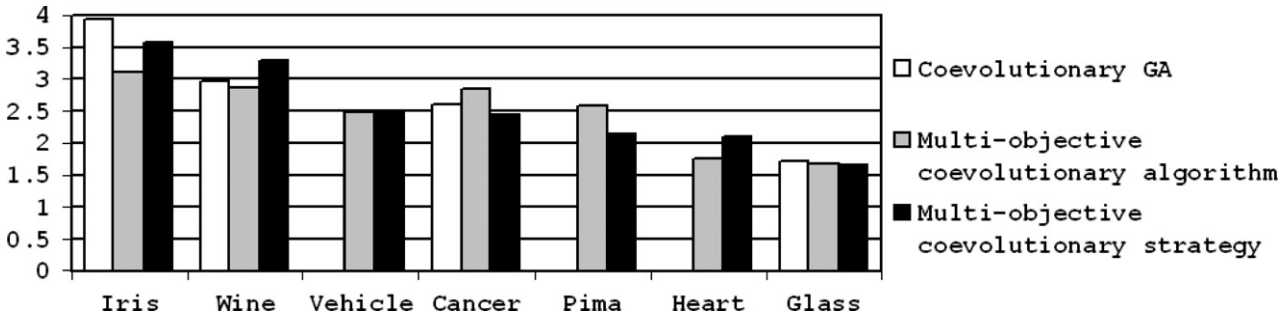

Fig. 6. Execution time reduction.

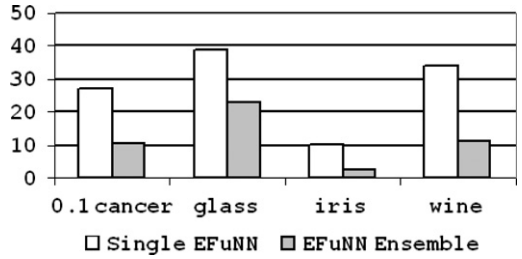

Fig. 7. Execution time average (in seconds) per database for co-evolutionary GA.

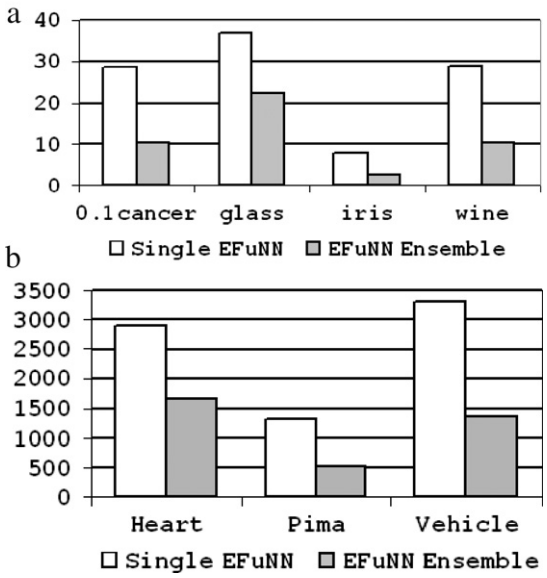

Fig. 8. Execution time average (in seconds) per database for multi-objective co-evolutionary GA.

Table 3
Measures related to the execution times (in seconds) for co-evolutionary GA

		Cancer	Glass	Iris	Wine
Ens	Av	104.5056	22.8528	2.6222	11.4722
	SD	60.1617	9.4891	0.4912	1.9970
	Min	38	9	2	7
	Max	320	39	4	16
Sing	Av	271	38.9361	10.3306	33.9139
	SD	150.4960	21.8213	4.9402	11.7227
	Min	105	11	4	15
	Max	724	71	20	54

6.3.1. Relation between execution time and rule nodes number

As was explained in Section 3, the neural networks of each species learn a reduced training set. In this way, they have less nodes than a single neural network used to learn the whole training set. So, each training example is presented to a relatively small neural network, reducing the training time. This section presents an analysis which shows the strong relation between the reduction in the number of nodes and the reduction in the execution time.

Two measures will be used. One of them is the percentage of execution time, which is the average execution time of CONE divided by the average execution time of the corresponding evolutionary algorithm. As the execution time of CONE in the experiments is always lower than the execution time of the corresponding evolutionary algorithms, this measure will always be between 0 and 1. The lower its value, the higher the reduction in the execution time when CONE is used.

The other measure is the percentage of nodes number, which is the average weighted nodes number of the ensembles divided by the average nodes number of the single EFuNNs. The weighted nodes number of an ensemble is:

$$WRN = \frac{\sum_{i=0}^{i<EN-1} (T_i \; RN_i)}{\sum_{i=0}^{i<EN-1} T_i}, \tag{9}$$

where RN_i is the number of rule nodes of EFuNN i of the ensemble, T_i is the number of training instances associated with the EFuNN i and EN is the ensemble size.

The experiments show that the weighted nodes number of an ensemble generated by CONE is always lower than the nodes number of a single EFuNN generated by a corresponding evolutionary algorithm, as expected. In this way, the percentage of nodes number is always between 0 and 1. As the percentage of execution time, the lower its value, the higher the reduction in the weighted rule nodes number when CONE is used.

Fig. 9 shows the percentage of execution time and the percentage of nodes number for each of the 3 different partitions of training data used in the experiments. It is necessary to calculate the measures for each one of the different partitions because they produce a different number of clusters of the input space and, consequently, a different ensemble size. It can be observed that a reduction/increase in the execution

Table 4
Measures related to the execution times (in seconds) for multi-objective co-evolutionary GA

		Cancer	Glass	Iris	Wine	Heart	Pima	Vehicle
Ens	Av	101.7556	22.2667	2.5444	10.2111	4018.2556	1628.5222	3401.9556
	SD	27.6626	4.2075	0.5228	1.1167	19.1073	26.3477	103.6192
	Min	55	12	2	8	3674	1276	1995
	Max	203	33	4	13	4443	1919	4167
Sing	Av	286.8889	37.1778	7.8778	29.0111	8406.4444	3465.2444	8343.3444
	SD	54.4691	7.5633	1.9707	4.9865	27.8621	3.3603	6.1166
	Min	186	20	5	20	8005	3398	8209
	Max	416	54	13	44	8917	3538	8481

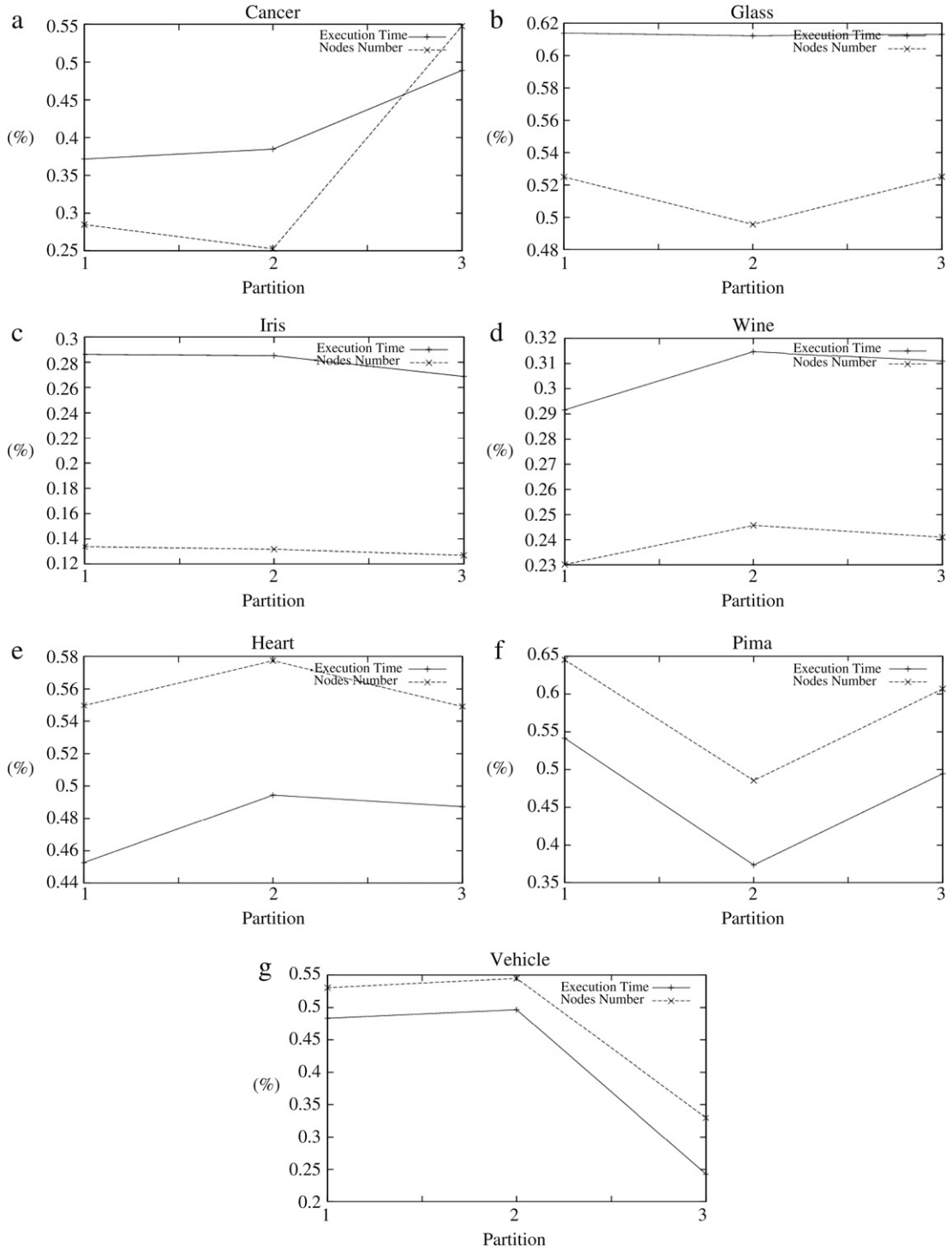

Fig. 9. Percentages of execution time and rule nodes.

time is related to a reduction/increase in the percentage of nodes number, showing the strong relation between them.

6.4. Accuracy

The experiments show that the accuracy of the ensembles created by using CONE is statistically equal or better than the accuracy of single EFuNNs created using a evolutionary algorithm corresponding to the co-evolutionary algorithm used with CONE. Besides, the weighted average combination method is always either statistically more accurate or equal to the arithmetic average.

Fig. 10 shows the classification error averages for the ensembles generated by CONE with the co-evolutionary strategy and the single EFuNNs generated by the corresponding evolutionary strategy. Observe that the classification error average of Glass database is multiplied by 0.1 in this figure. Table 5 shows the classification error averages, standard

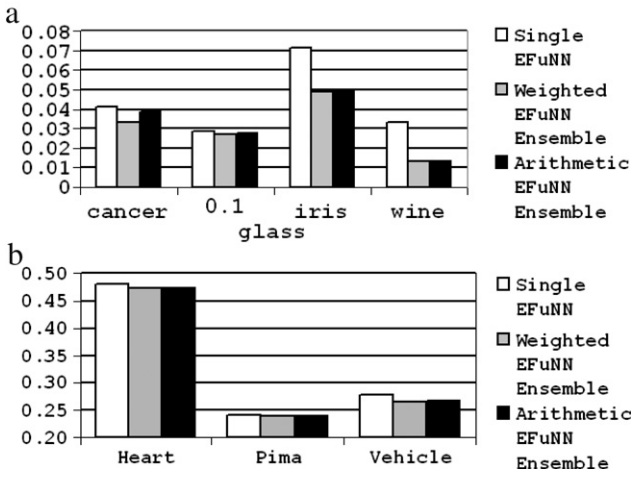

Fig. 10. Classification error average per database for multi-objective co-evolutionary strategy.

Table 5
Measures related to the classification errors for multi-objective co-evolutionary strategy

		Cancer	Glass	Iris	Wine	Heart	Pima	Vehicle
W Ens	Av	0.0332	0.2673	0.0490	0.0133	0.4722	0.2394	0.2661
	SD	0.0104	0.0340	0.0339	0.0182	0.0034	0.0021	0.0016
	Min	0.0172	0.1698	0.0256	0	0.4190	0.2031	0.2347
	Max	0.0632	0.3774	0.1026	0.0667	0.5333	0.3021	0.3052
A Ens	Av	0.0386	0.2736	0.0490	0.0133	0.4722	0.2389	0.2673
	SD	0.0138	0.0414	0.0339	0.0182	0.0034	0.0021	0.0016
	Min	0.0172	0.1698	0.0256	0	0.4190	0.2031	0.2394
	Max	0.0805	0.3962	0.1026	0.0667	0.5333	0.3021	0.3052
Sing	Av	0.0414	0.2845	0.0712	0.0333	0.4815	0.2399	0.2776
	SD	0.0127	0.0415	0.0330	0.0214	0.0028	0.0024	0.0016
	Min	0.0172	0.2075	0	0	0.4333	0.1771	0.2394
	Max	0.0862	0.3962	0.1282	0.0889	0.5523	0.2969	0.3099

Table 6
P-Value of T Student Test comparing the classification error averages for multi-objective co-evolutionary strategy

	Cancer	Glass	Iris	Wine	Heart	Pima	Vehicle
W Ens x A Ens	**0.0000**	**0.0000**	=	=	=	0.1172	**0.0010**
W Ens x Sing	**0.0000**	**0.0027**	**0.0000**	**0.0000**	**0.0003**	0.8414	**0.0000**
A Ens x Sing	**0.0000**	0.0726	**0.0000**	**0.0000**	**0.0003**	0.7066	**0.0000**

deviations, minimal and maximum values. In this table and in the other tables of this paper, "W Ens" means Weighted EFuNN ensemble (which uses weighted combination method), "A Ens" means Arithmetic EFuNN ensemble (which uses arithmetic combination method) and "Sing" means Single EFuNN. Table 6 shows the p-values of the paired T Student tests performed to prove the analyses made with the classification errors. The signals "=" indicate that the classification errors of all the paired executions were equal and it is not possible to calculate the p-value. The p-values that are less than 0.05 are emphasized in bold and indicate that there is a difference between the compared algorithms.

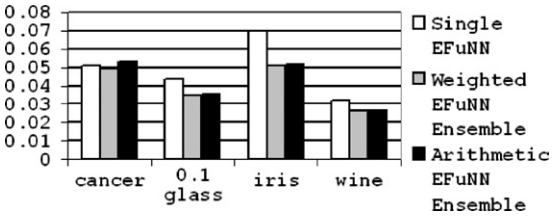

Fig. 11. Classification error average per database for co-evolutionary GA.

Table 7
Measures related to the classification errors for co-evolutionary GA

		Cancer	Glass	Iris	Wine
W Ens	Av	0.0492	0.3495	0.0512	0.0267
	SD	0.0196	0.1101	0.0370	0.0275
	Min	0.0115	0.1698	0.0256	0.0000
	Max	0.1207	0.6604	0.1282	0.1333
A Ens	Av	0.0531	0.3507	0.0515	0.0267
	SD	0.0244	0.1094	0.0371	0.0276
	Min	0.0057	0.1509	0.0256	0.0000
	Max	0.1379	0.6604	0.1282	0.1333
Sing	Av	0.0512	0.4340	0.0698	0.0316
	SD	0.0240	0.1648	0.0377	0.0209
	Min	0.0057	0.1698	0.0000	0.0000
	Max	0.1379	0.8868	0.2051	0.1111

Table 5 in combination with Table 6 shows that the classification error averages of the weighted ensembles were considered statistically lower than the classification error averages of the single EFuNNs for all databases but Pima. For Pima, the average of the ensembles and single EFuNNs was considered statistically equal. The classification error averages of the arithmetic ensembles were considered statistically lower than the classification error averages of the single EFuNNs for all databases but Glass and Pima. For Glass and Pima, the average of the ensembles and single EFuNNs was considered statistically equal. Notice that the p-values are very low to almost all databases.

For Cancer, Glass and Vehicle databases, the classification error averages of the weighted EFuNN ensembles were considered statistically lower than the classification error averages of the arithmetic EFuNN ensembles. For Wine, Iris, Heart and Pima, the averages were considered statistically equal.

For the executions with the other co-evolutionary algorithms, the accuracy of ensembles generated by using CONE was also always either equal or better than the accuracy of single EFuNNs generated by the corresponding evolutionary algorithms. The only exception was when the multi-objective co-evolutionary genetic algorithm was used with Cancer database and arithmetic average of the EFuNNs' outputs. Although, this is a database in which usually a good accuracy is attained by learning methods and it is known that ensembles are more helpful when used to solve difficult problems.

Figs. 11 and 12 show the classification error averages for the other co-evolutionary algorithms. Tables 7 and 8 show the classification error averages, standard deviations, minimal and maximum values for these algorithms and Tables 9 and 10 show the p-values of the T Student tests.

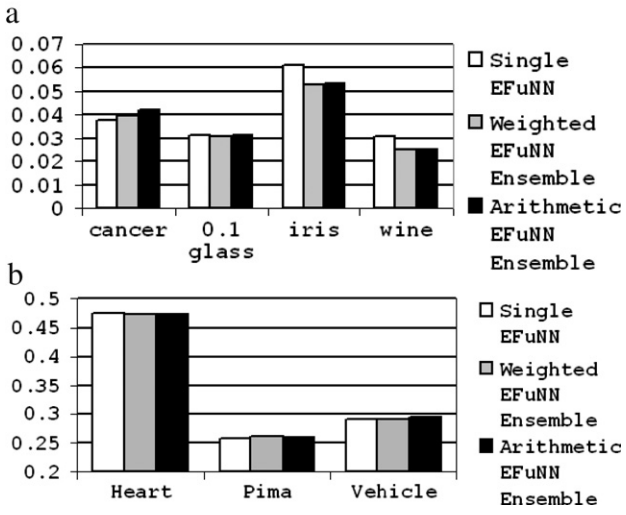

Fig. 12. Classification error average per database for multi-objective co-evolutionary GA.

Table 8
Measures related to the classification errors for multi-objective co-evolutionary GA

		Cancer	Glass	Iris	Wine	Heart	Pima	Vehicle
W Ens	Av	0.0398	0.3096	0.0530	0.0254	0.4727	0.2614	0.2917
	SD	0.0132	0.0611	0.0397	0.0239	0.0031	0.0029	0.0027
	Min	0.0172	0.1698	0.0256	0.0000	0.4190	0.1771	0.2441
	Max	0.0805	0.4340	0.1282	0.0889	0.5381	0.3438	0.3897
A Ens	Av	0.0424	0.3122	0.0536	0.0254	0.4726	0.2608	0.2947
	SD	0.0151	0.0647	0.0395	0.0239	0.0031	0.0029	0.0026
	Min	0.0172	0.1698	0.0256	0.0000	0.4190	0.1823	0.2441
	Max	0.0862	0.4340	0.1282	0.0889	0.5381	0.3438	0.3803
Sing	Av	0.0375	0.3157	0.0612	0.0311	0.4744	0.2577	0.2907
	SD	0.0130	0.0740	0.0337	0.0213	0.0027	0.0027	0.0024
	Min	0.0057	0.1132	0.0000	0.0000	0.4238	0.1927	0.2441
	Max	0.0632	0.5660	0.1795	0.1333	0.5333	0.3177	0.3615

Table 9
p-value of T Student test comparing the classification error averages for co-evolutionary GA

	Cancer	Glass	Iris	Wine
W Ens x A Ens	**0.0000**	**0.0283**	**0.0453**	0.3180
W Ens x Sing	0.1176	**0.0000**	**0.0000**	**0.0048**
A Ens x Sing	0.1629	**0.0000**	**0.0000**	**0.0054**

Table 10
p-value of T Student test comparing the classification error averages for multi-objective co-evolutionary GA

	Cancer	Glass	Iris	Wine	Heart	Pima	Vehicle
W Ens x A Ens	**0.0076**	**0.0331**	0.1585	=	0.3200	0.1051	**0.0000**
W Ens x Sing	0.2415	0.4948	**0.0495**	0.1276	0.4751	0.3120	0.8078
A Ens x Sing	**0.0182**	0.6984	0.0627	0.1276	0.4627	0.3933	0.2887

6.5. Multi-objective co-evolutionary GA versus multi-objective co-evolutionary strategy

All the results presented in this section correspond to executions which have used either CONE with the multi-objective co-evolutionary GA presented in the Section 5.2 or CONE with the multi-objective co-evolutionary strategy presented in the Section 5.3. The comparisons made in this section show that by changing the co-evolutionary algorithm used with CONE it is possible to attain different classification rates and execution times, although the main properties of CONE are the same (Sections 6.3 and 6.4). In this way, it is important to choose the co-evolutionary algorithm according to the requirements of the problem. For instance, the multi-objective co-evolutionary GA could be used when the emphasis is the reduction in the execution time. When the accuracy is more important, the multi-objective co-evolutionary strategy presented in the Section 5.3 could be used.

Table 11
p-value of T Student test comparing the classification error averages for multi-objective co-evolutionary strategy and multi-objective co-evolutionary GA

	Cancer	Glass	Iris	Wine	Heart	Pima	Vehicle
Weig Ens Multi-obj ES x GA	**0.0002**	**0.0000**	0.4695	**0.0002**	0.9271	**0.0000**	**0.0000**
Arit Ens Multi-obj ES x GA	0.0773	**0.0000**	0.4071	**0.0002**	0.9362	**0.0000**	**0.0000**

Fig. 13 shows the classification error averages of the EFuNN ensembles using multi-objective co-evolutionary GA and multi-objective co-evolutionary strategy. Table 11 shows the p-values of the not-paired T student tests made to prove the analyses made in this section.

It can be observed that the classification error average of the ensembles created using multi-objective co-evolutionary strategy are statistically equal or lower than the averages of the ensembles created using multi-objective co-evolutionary GA. However, as shown by Fig. 14 and the p-values of the statistic tests made to compare the execution time averages (which were all 0.0000), the execution time using multi-objective co-evolutionary strategy is always statistically higher than the one using multi-objective co-evolutionary GA. In this way, it is important to choose the co-evolutionary algorithm according to the requirements of the problem.

7. Conclusion

This paper introduces CONE, which is an approach to create neural network ensembles. It creates the ensembles in an innovative way, by explicitly partitioning the input space through a clustering method. In this way, the neural networks that compose the ensemble specialize in different parts of the problem space and work in a divide-and-conquer manner. The clustering of the input space also allows a reduction in the number of nodes of the neural networks that compose the ensemble, thus reducing the execution time of the learning process. By using the approach in a distributed way, it would be possible to further reduce the execution time. A co-evolutionary algorithm can be used to optimize the architecture of the neural networks that compose the ensemble.

The experiments show that CONE indeed considerably reduces the execution time of the learning process even when a distributed implementation is not used. Besides, the accuracy

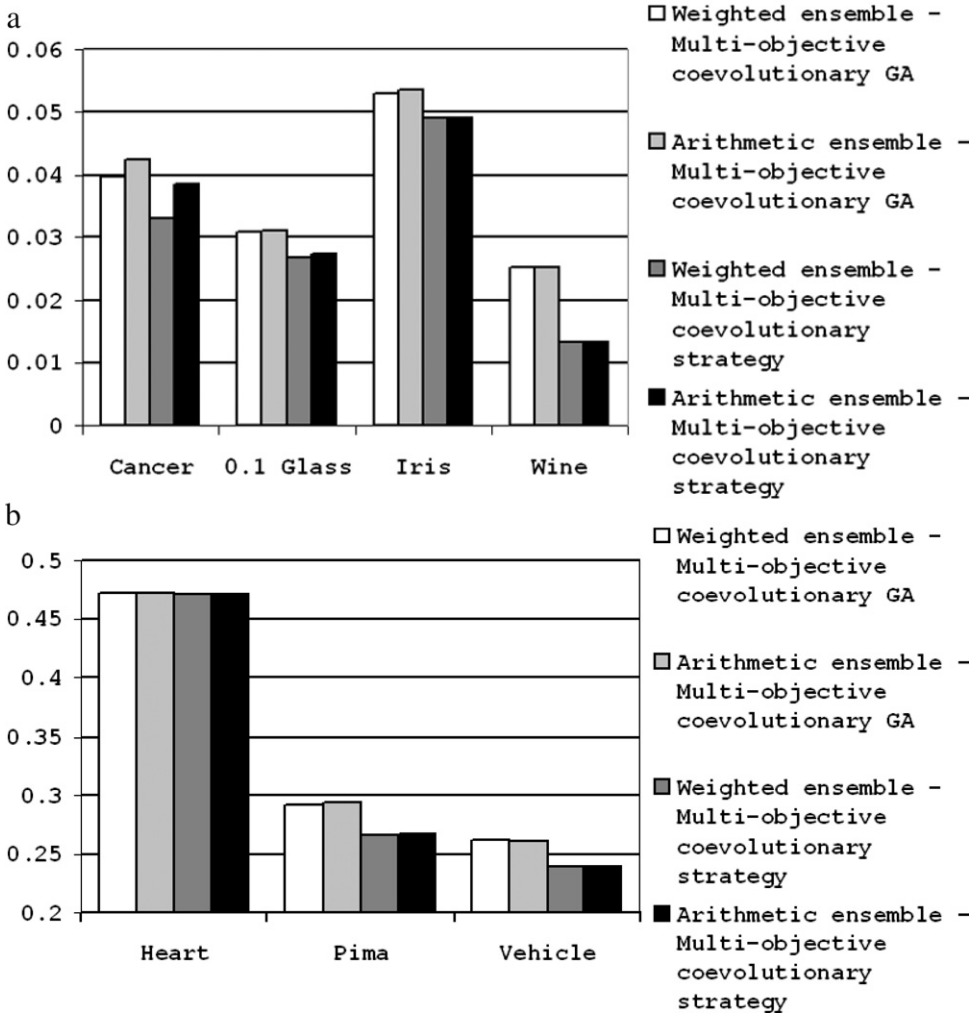

Fig. 13. Classification error average per database for multi-objective co-evolutionary GA and multi-objective co-evolutionary strategy.

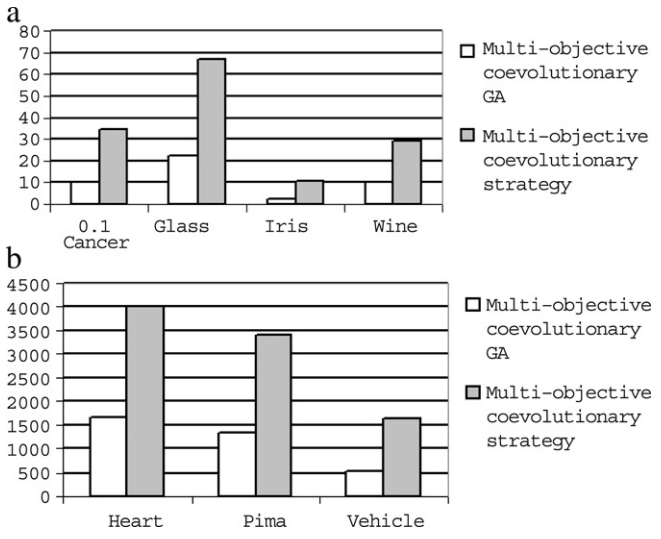

Fig. 14. Execution time average (in seconds) per database for multi-objective co-evolutionary GA and multi-objective co-evolutionary strategy.

of the ensembles generated by CONE is similar or higher than the accuracy of a single EFuNN created using an evolutionary algorithm. The experiments used 3 different co-evolutionary algorithms, showing that these results are obtained even using different co-evolutionary algorithms. However, it is important to notice that different co-evolutionary algorithms are more appropriate to different problem purposes. For example, when accuracy is more important, a multi-objective co-evolutionary strategy (Section 5.2) could be used. When the execution time is more important, a multi-objective co-evolutionary genetic algorithm (Section 5.3) could be used. The experiments also reveal that the weighted average combination method has always higher or equal accuracy to the arithmetic average.

Future work includes the analysis of the effect of other co-evolutionary algorithms with CONE, the use of more than one representative of each species to compose the ensemble of neural networks, the execution of CONE in a distributed way, the extraction of rules explaining the ensemble's working and the analysis of CONE's behaviour in on-line mode.

Acknowledgments

The first author would like to thank the Federal University of Pernambuco (Brazil), the Brazilian Institute of Technology for Development (LACTEC), the Brazilian Technological and Scientific Development Council (CNPq) and the University

of Birmingham. The authors are grateful to the anonymous referees for their valuable comments, which have helped to improve the quality of this paper.

Appendix A. Clustering method

The clustering algorithm used with CONE in this paper was the Evolving Clustering Method (ECM) (Kasabov, 2001a). The Algorithm A.1 presents it.

In this paper, the distance between two vectors x and y denotes the General Euclidean Distance, defined as follows:

$$\|x - y\| = \sqrt{\frac{\sum_{0}^{size-1} (x_i - y_i)^2}{size}}. \quad \text{(A.1)}$$

Algorithm A.1 (*Clustering Method*). Let *NumEx* be the number of patterns and *Dthr* be a distance threshold.

(i) Create the first cluster C_0 by simply taking the position of the first pattern as the first cluster center Cc_0 and setting a value 0 for its cluster radius Ru_0.
(ii) For each input pattern x_i from $i = 1$ to $NumEx - 1$ do:
  (a) Determine the distance between x_i and all N cluster centers Cc_j already created:
  $D_{ij} = \|x_i - Cc_j\|,\ j = 0, 1, \ldots, N - 1$.
  (b) If there is a distance value $D_{ij} \leq Ru_j$, it means that x_i belongs this cluster. In this case, neither a new cluster is created nor an existing cluster is updated.
  (c) Else
    (i) Find the cluster C_α with the minimum distance $D_{i\alpha} = \|x_i - Cc_\alpha\| = min(\|x_i - Cc_j\|),\ j = 0, 1, \ldots, N - 1$.
    (ii) If $D_{i\alpha} > Dthr$, create a new cluster, in the same way as described in the step (i).
    (iii) Else update C_α: increment the number of patterns accommodated by C_α ($NExs_\alpha = NExs_\alpha + 1$); update Cc_α ($Cc_\alpha = Cc_\alpha + (x_i - Cc_\alpha)/NExs_\alpha$) and make R_α be the maximum between the following values: 1. the distance between the old Cc_α and the new Cc_α plus the old Ru_α and 2. the distance between x_i and the new center Cc_α.

Appendix B. EFuNNs

This section describes the EFuNN learning procedure. It is recommended to read Kasabov (2001b) for further details.

EFuNNs are a class of ECoSs which join the neural networks functional characteristics to the expressive power of fuzzy logic. They have a five-layer architecture. The first layer represents the input vector, the second represents the fuzzy quantification of the input vector, the third represents the associations between fuzzy input space and fuzzy output space, the fourth represents the fuzzy quantification of the output vector and the fifth represents the output vector.

Learning occurs at the rule nodes layer. Each node *rj* of this layer is represented by two vectors of connection weights (*W1(rj)* and *W2(rj)*). *W1* represents the coordinates of the nodes in the fuzzy input space and it is adjusted through unsupervised learning. *W2* represents the coordinates of the nodes in the fuzzy output space and it is adjusted through supervised learning. The learning rules are the following:

$$W1(r_j(t+1)) = W1(r_j(t)) + lr1(r_j(t)) * (x_f - W1(r_j(t))) \quad \text{(B.1)}$$

$$W2(r_j(t+1)) = W2(r_j(t)) + lr2(r_j(t)) * (y_f - A2) * A1(r_j(t)), \quad \text{(B.2)}$$

where: x_f and y_f are the fuzzy input and fuzzy output vectors; $lr1(r_j(t))$ and $lr2(r_j(t))$ are the learning rates for the $W1$ and $W2$ weights of the node r_j at time t; $A2$ is the fuzzy output activation vector and $A1(r_j(t))$ is the activation value of the rule node r_j at time t.

The EFuNN learning algorithm is briefly described below.

Algorithm B.1 (*EFuNN Learning Algorithm*).

(i) Set initial values for the following system parameters: number of membership functions; initial sensitivity threshold S of the nodes (it is also used to determine the initial radius of the receptive field of a node r_j, when it is created ($R(r_j) = 1 - S$); error threshold E; aggregation parameter *Nagg*; pruning parameters *OLD* and Pr; *m-of-n* value (number of highest activation nodes used in the learning); maximum radius of the receptive field *Mrad*; rule extraction thresholds $T1$ and $T2$.
(ii) Set the first rule node r_0 to memorize the first example (x, y):
  $W1(r_0) = x_f \quad \text{and} \quad W2(r_0) = y_f$,
  where x_f and y_f are the vectors of fuzzy quantification of the vectors x and y, respectively.
(iii) Repeat for each new input-output pair (x, y) presentation:
  (a) Determine the local normalized fuzzy distance D between x_f and the $W1$ weights. The distance D between two fuzzy vectors $x1$ and $x2$ is calculated as follows:
  $D(x1, x2) = subabs(x1, x2)/sumabs(x1, x2)$,
  where $subabs(x1, x2)$ is the sum of all absolute values of the vector obtained after subtraction of the fuzzy vectors $x1$ and $x2$ and $sumabs(x1, x2)$ is the sum of all absolute values of the vector obtained after sum of the fuzzy vectors $x1$ and $x2$.
  (b) Calculate the activations $A1$ of all rule nodes. An example of how it can be calculated is:
  $A1 = 1 - D(W1(r_j), x_f)$.
  (c) Select the rule node r_k that has the smallest distance $D(W1(r_k), x_f)$ and that has activation $A1(r_k) >= S(r_k)$. In the case of *m-of-n* learning, select m nodes instead of just one node.
  (d) If this node does not exist
    (i) Create a new rule node for (x_f, y_f).

(e) Else
  (i) Determine the activation $A2$ of the output layer and the normalized output error $Err = subabs(y, y')/Nout$, where y is the desired output, y' is the obtained output and *Nout* is the number of nodes of the output layer.
  (ii) If $Err > E$, create a new rule node for (x_f, y_f).
  (iii) Else, apply the learning rules to $W1(r_k)$ and $W2(r_k)$ (in the case of *m-of-n* learning, the rules are applied to the m rule nodes).

(f) Apply the aggregation procedure after the presentation of Nagg examples.

(g) Update the parameters $S(r_k)$, $R(r_k)$, $Age(r_k)$ and $TA(r_k)$. $TA(r_k)$ can be, for example, the sum of the activations $A1$ obtained for all examples that r_k accommodates.

(h) Prune rule nodes, if necessary, according to OLD and Pr.

(i) Extract rules, according to $T1$ and $T2$.

References

Chandra, A., & Yao, X. (2006). Ensemble learning using multi-objective evolutionary algorithms. *Journal of Mathematical Modelling and Algorithms*, *5*(4), 417–445.

Chan, Z. S. H., & Kasabov, N. (2004). Evolutionary computation for on-line and off-line parameter tuning of evolving fuzzy neural networks. *International Journal of Computational Intelligence and Applications*, *4*(3), 309–319.

Chen, H., & Yao, X. (2006). Evolutionary multiobjective ensemble learning based on bayesian feature selection. In *Proceedings of the 2006 IEEE congress on evolutionary computation* (pp. 971–978).

Chen, H., & Yao, X. (2007). Evolutionary random neural ensembles based on negative correlation learning. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation* (pp. 1468–1474).

Dietterich, T. G. (1998). Machine-learning research: Four current directions. *The AI Magazine*, *18*(4), 97–136.

Duell, P., Fermin, I., & Yao, X. (2006). Speciation techniques in evolved ensembles with negative correlation learning. In *Proceedings of the 2006 IEEE congress on evolutionary computation* (pp. 11086–11090).

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. Berlin: Springer.

Fonseca, C. M., & Fleming, P. J. (1998). Multi-objective optimization and multiple constraint handling with evolutionary algorithms — Part I: A unified formulation. *IEEE Transactions on Systems, Man and Cybernetics — Part A*, *28*(1), 26–37.

Garcia-Pedrajas, N., Hervas-Martinez, C., & Ortiz-Boyer, D. (2005). Cooperative coevolution of artificial neural network ensembles for pattern classification. *IEEE Transactions on Evolutionary Computation*, *9*(3), 271–302.

Inoue, H., & Narihisa, H. (2005). Self-organizing neural grove and its applications. In *Proceedings of the 2005 international joint conference on neural networks*, *2* (pp. 1205–1210).

Islam, M. M., Yao, X., & Murase, K. (2003). A construtive algorithm for training cooperative neural network ensembles. *IEEE Transactions on Neural Networks*, *14*(4), 820–834.

Jacobs, R. A., & Jordan, M. I. (1991). Adaptive mixture of local experts. *Neural Computation*, *3*, 79–87.

Jordan, M. I., & Jacobs, R. A. (1992). Hierarchies of adaptive experts. In *Advances in neural information processing systems, NIPS'91, Part XIII*: *Vol. 4* (pp. 985–992). Morgan-Kaufmann.

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*, 181–214.

Kasabov, N. (2001a). Ensembles of EFuNNs: An architecture for a multimodule classifier. In *Proceedings of the international conference on fuzzy systems*, *3*, 1573–1576.

Kasabov, N. (2001b). Evolving fuzzy neural networks for supervised/unsupervised on-line, knowledge-based learning. *IEEE Transactions on Systems, Man and Cybernetics*, *31*(6), 902–918.

Kasabov, N. (2003). *Evolving connectionist systems*. Great Britain: Springer.

Kasabov, N. (2007). *Evolving connectionist systems: The knowledge engineering approach* (2nd ed.). London: Springer.

Kasabov, N., Song, Q., & Nishikawa, I. (2003). Evolutionary computation for dynamic parameter optimization of evolving connectionist systems for on-line prediction of time series with changing dynamics. In *IJCNN'2003*, *1* (pp. 438–443).

Li, K., Huang, H.-K., Ye, X.-C., & Cui, L.-J. (2004). A selective approach to neural network ensemble based on clustering technology. In *Proceedings of the 3rd international conference on machine learning and cybernetics* (pp. 3229–3233).

Liao, X., Li, H., & Carin, L. (2007). Quadratically gated mixture of experts for incomplete data classification. In *Proceedings of the 24th international conference on machine learning* (pp. 553–560).

Liu, Y., & Yao, X. (1998). Towards designing neural network ensembles by evolution. In *Lecture notes in computer science*: *Vol. 1498*. *In Proceedings of the fifth international conference on parallel problem solving from nature* (pp. 623–632). Berlin: Springer-Verlag.

Liu, Y., Yao, X., & Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, *4*(4), 380–387.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observation. In J. Neyman, et al., (Eds.), *Proceedings of the 5th Berkely symposium on mathematical statistics and probability*: *Vol. 1* (pp. 281–297). Berkely: U. California Press.

Minku, F. L., & Ludermir, T. B. (2005). Evolutionary strategies and genetic algorithms for dynamic parameter optimization of evolving fuzzy neural networks. In *CEC'2005*, *3* (pp. 1951–1958).

Minku, F. L., & Ludermir, T. B. (2006a). EFuNN ensembles construction using CONE with multi-objective GA. In *Proceedings of the ninth Brazilian symposium on neural networks* (pp. 9–14). Ribeirao Preto, Brazil: IEEE Press.

Minku, F. L., & Ludermir, T. B. (2006b). EFuNNs ensembles construction using a clustering method and a coevolutionary genetic algorithm. In *CEC'2006* (pp. 5548–5555).

Newman, D., Hettich, S., Blake, C., & Merz, C. (1998). UCI Repository of machine learning databases.

Potter, M. A., & De Jong, K. A. (2000). Cooperative coevolution: An architeture for evolving coadapted subcomponents. *Evolutionary Computation*, *8*(1), 1–29.

Prechelt, L. (1994). PROBEN1 — A set of neural network benchmark problems and benchmarking rules. *Tech. rep. 21/94*. Fakultät für Informatik. Universität Karlsruhe.

Ranawana, R., & Palade, V. (2005). A neural network based multi-classifier system for gene identification in DNA sequences. *Neural Computing and Applications*, *14*(2), 122–131.

Song, Q., & Kasabov, N. (2002). DENFIS: Dynamic evolving neural-fuzzy inference system and its application to time-series prediction. *IEEE Transactions on Fuzzy Systems*, *10*(2), 144–154.

Watts, M., & Kasabov, N. (2001). Dynamic optimisation of evolving connectionist system training parameters by pseudo-evolution strategy. In *CEC'2001*, *2* (pp. 1335–1342).

Watts, M., & Kasabov, N. (2002). *CEC'2002*: *Vol. 1*. *Evolutionary optimisation of evolving connectionist systems* (pp. 606–610). Honolulu, Hawaii: IEEE Press.

Witten, I. H., & Frank, E. (2000). *Data mining — pratical machine learning tools and techniques with java implementations*. San Francisco: Morgan Kaufmann Publishers.

Yao, X., & Liu, Y. (1998). Making use of population information in evolutionary artificial neural networks. *IEEE Transactions on Systems, Man and Cybernetics — Part B*, *28*(3), 417–425.