SimBa: A novel similarity-based crossover for neuro-evolution

Mauro Dragoni, Antonia Azzini, Andrea G. B. Tettamanzi

SimBa: A Novel Similarity-Based Crossover for Neuro-Evolution

Mauro Dragoni[1], Antonia Azzini[2], Andrea G. B. Tettamanzi[2]

[1]) *Fondazione Bruno Kessler (FBK-IRST)*
Via Sommarive 18, Povo (Trento), Italy
dragoni@fbk.eu
[2]) *Università degli Studi di Milano, DTI*
Via Bramante 65, 26013 Crema (CR), Italy
antonia.azzini@unimi.it, andrea.tettamanzi@unimi.it

Abstract

This work presents the `SimBa` (for Similarity-Based) crossover, a novel crossover operator specifically designed for the evolutionary optimization of neural network topologies that aims at overcoming one of the major problems generally related to the crossover operator, known as the permutation problem.

The `SimBa` crossover starts by looking for a *local similarity* between two individuals selected from the population. The contribution of each neuron of the layer selected for the crossover is computed, and the neurons of each layer are reordered according to their contribution. Then, each neuron of the layer in the first individual is associated with the most *similar* neuron of the layer in the other individual, and the neurons of the layer of the second individual are re-ranked by considering the associations with the neurons of the first one. Finally, the neurons above a randomly selected cut-point are swapped to generate the offspring of the selected individuals.

An approach exploiting this operator has been implemented and applied to six well-known benchmark classification problems. The experimental results, compared to those obtained by other techniques, show how this new crossover operator can help produce compact neural networks with satisfactory generalization capability and accuracy.

Keywords: Artificial Neural Networks, Evolutionary Algorithms, Crossover Operator.

1. Introduction

Artificial Neural Networks (ANNs) and Evolutionary Algorithms (EAs) are two important members of the family of computational methods collectively known as *computational intelligence*. Their advantages over conventional methods [1], like their conceptual and computational simplicity and their applicability to broad classes of optimization tasks, make them very attractive to approach those problems that pose difficulties to traditional techniques [2, 3, 4].

The success of an ANN application usually requires a high number of experiments and several parameters affect, during the design, how easy a solution is to find. Among them, particular attention has to be given to those related to the architectural design of the neural network. As a matter of fact, one of the problems to which EAs have been successfully applied is ANN design. Approaches using ANNs designed and/or optimized by means of an EA may be called Evolutionary Artificial Neural Networks (EANNs) [5] or, according to a more recent proposal, Neuro-Evolutionary

approaches [2, 6]. EAs are able to overcome an important limitation of traditional neural network learning, namely that it may get trapped in local minima.

Among the genetic operators that can be applied during the evolutionary process, some authors have regarded crossover as inefficient, due to the so-called permutation (or competing conventions) problem [7, 8]. This problem occurs because the same network can be genetically represented by many and different encodings. This problem is also indicated as a many-to-one mapping from the representation of the solutions (the genotype) to the actual ANNs (the phenotype) [5].

Nevertheless, many successful applications of neuro-evolution using crossover have been reported: Hancock, for example, conducted studies on structural optimization [7], Garcia-Pedrajas and colleagues have recently investigated a combination of structure and weight evolution [9], and it is worth emphasizing that none of them has found any significant detrimental effects attributable to the permutation problem. Accordingly, there is a need to re-evaluate the traditional theoretical claims with regard to this problem.

The aim of this article is to provide an organic description and discussion of a crossover operator, called `SimBa`, recently proposed by the authors (see [10] for an initial formulation, which has then been improved in [11]) to solve the permutation problem. The operator exploits the concept of similarity among individuals, which is one of the first ideas developed in the literature for solving such problem. The `SimBa` operator has been implemented within a previously published neuro-evolutionary approach for feedforward neural network design [12], based on the simultaneous optimization of the network topology and of the connection weights, and successfully tested on a number of benchmark classification problems. As a matter of fact, even if they have found other applications, e.g., in robotics, neuro-evolutionary approaches may still be counted among the most useful approaches to classification according to the recent literature.

The rest of the paper is organized as follows: Section 2 describes the problem, introducing the features and the main critical aspects of the crossover, while a brief description about the ANN optimization is presented in Section 3. The neuro-evolutionary approach used in this work is then summarized in Section 4, while a more detailed description of the implemented crossover is given in Section 5. All the experiments are then presented, compared, and discussed in Section 6. Section 7 reports some final remarks.

2. Problem Description

Among the many applications presented in the literature in the area of neuro-evolution, different approaches show interesting combinations of network architecture and weight optimization, carried out simultaneously. In fact, the choice of an ANN structure has a considerable impact on the processing power and learning capability of the classifier. In ANN evolution, when the recombination process is considered, the crossover operator exchanges the architectures of individuals in the population, identified as parents, in order to search for better solutions. However, crossover can be seriously disruptive when applied, for examples, to ANNs whose genotypes are incompatible, even though their phenotypes might be indistinguishable: this is called the *permutation problem* or the *competing conventions problem*. For this reason, some authors prefer to avoid using a crossover operator and only apply mutation for neural network evolution. Several implementations have been reported in the literature, mainly using Gaussian [13, 14] or Cauchy [5] mutation as the main search operator. Froese and Spier proposed the so-called 'convergence argument' [15] to counter some critiques in the literature claiming that the crossover is generally very difficult to apply, since it tends to destroy feature detectors found by the global evolutionary process while searching for

the best individual in a population [16, 17]. The argument mantains that crossover is usually not harmful in practice, because, for most generations of an evolutionary run, the population will have converged into an area of the genotypic search space which it continues to explore.

As a countermeasure, some authors proposed to prevent competing conventions by matching pairs of neurons of mated solutions according to their similarity prior to the crossover operation [18]. Alternatively, sub-populations (species) of neurons may be evolved, each of them corresponding to a position in a pre-defined ANN structure [19]. Along these lines, some authors proposed new crossover operators and representations [20], some others restricted the use of crossover for the topology or weight evolution, for example, by applying graph-matching techniques to non-fixed structures [15, 21]. Mandischer [22] used, for example, a simple crossover operator, similar to the naive crossover that has been used in this work for benchmarking the `SimBa` crossover, while Miikkulainen and colleagues implemented another interesting approach, the NeuroEvolution of Augmenting Topologies (NEAT) method [2], which has become very popular in recent years. NEAT includes a method of gene tracking, called historical marking, which avoids competing conventions and prevents incompatible individuals from mating.

It should be noted that the 'convergence argument' cited above is not a good reason to give up the quest for a suitable crossover operator for ANN evolution. After all, in general, recombination in EAs is most useful *before* convergence, during the exploration phase, when it may help locating promising areas of the search space. This is exactly why the local-similarity based crossover presented in this work turns out to be beneficial.

We have given particular attention to two empirical studies [23, 7], which focus on the evolution of the single network unit involved in the crossover operator, the hidden node. Their aim was to emphasize the equivalence between hidden nodes of ANNs, in order to identify similarly performing units prior to crossover, avoiding all the disruptive effects stated above. Following such an idea, we extend our neuro-genetic approach already presented in the literature [24, 12], which implements a joint optimization of weights and network structure, by defining a novel crossover operator. This operator allows recombination of individuals that have different topologies, but with hidden nodes that are similarly performing in the cutting point of the hidden layer randomly chosen (indicated in the approach as *local similarity*). The evolutionary process does not consider only a part, but complete multilayer perceptrons (MLPs), achieving satisfactory performances and generalization capabilities, as well as reduced computational costs and network sizes.

3. Neuro-Evolutionary Classifier Systems

Generally speaking, a supervised ANN is composed of simple computing units (the *neurons*) which are connected to form a network [25, 26, 27]. Whether a neuron a influences another neuron b or not depends on the ANN structure. The extent of such influence, when there is one, depends on the weight assigned to each connection among the neurons. It is very difficult to find an optimal network (structure and weights) for a given problem.

Even though some authors do not consider supervised classification as a good domain for neuroevolution, preferring alternatives as support vector machines, Bayesian methods, or analytic optimization methods, neural networks are nevertheless one of the most popular tools for classification. The recent vast research activities in neural classification establish that neural networks are a promising alternative to various conventional classification methods and a large number of successful applications presented in the recent literature demonstrate that ANN design can be further improved by synergetically combining it with evolutionary algorithms, being able to take into

account all aspects of ANN design at once [17].

The review by Zhang [28], which provides a summary of the most important advances in classification with ANNs, makes it clear that the advantages of neural networks lie in different aspects: their capability to adapt themselves to the data without any explicit specification of functional or distributional form for the underlying model; they are universal functional approximators; they represent nonlinear and flexible solutions for modeling real world complex relationships; and, finally, they are able to provide a basis for establishing classification rules and performing statistical analysis. On the other hand, different neuro-evolutionary approaches have been successfully applied to a variety of benchmark problems and real-world classification tasks [29, 30, 31, 32, 33, 3]. Our neuro-evolutionary algorithm, too, has been already tested and applied with success to several real-world problems, showing how such an approach can be useful in different classification problems, like financial time series modeling [34], automated trading strategy optimization [24, 35], incipient fault diagnosis in electrical drives [36], automated diagnosis of skin diseases [37], brain-wave analysis [38], etc. Further insights on the evolutionary optimization of ANNs can be found in some broad surveys on the topic [6, 39, 4, 17].

4. The Neuro-Evolutionary Approach

The evolutionary process handles the design optimization of a population of ANNs with respect to a particular problem, in which all the available information is given as input to the neural networks. In this approach, no expert knowledge of the problem is required, since the evolutionary algorithm is able to automatically find the best solution for that particular problem.

The overall algorithm is based on the joint optimization of structure and weights, here briefly summarized, and the error backpropagation algorithm (BP) is used in the network learning phase; a more complete and detailed description can be found in [12]. An *indirect encoding* of ANNs is used and a *genotype* is 'decoded' into a *phenotype* ANN by the BP algorithm. Accordingly, it is the genotype which undergoes the genetic operators and which reproduces itself, whereas the phenotype is used *only* for calculating the genotype's fitness. The rationale for this choice is that the alternative of using BP, applied to the genotype, as a kind of 'intelligent' mutation operator, would boost exploitation while impairing exploration, thus making the algorithm too prone to being trapped in local optima. An example of an individual genotype is represented in Figure 1.

Thanks to this encoding, individual ANNs are not constrained to a pre-established topology. Unlike NEAT [2], which starts with minimal network topologies and then applies evolutionary mechanisms to augment them, our approach randomly initializes the network's population with different hidden layer sizes and different numbers of neurons for each individual according to two exponential distributions, in order not to constrain search and to provide a balanced mix of topologies. Network size is not bounded in advance, even though the fitness function may penalize large networks. The number of neurons in each hidden layer is constrained to be greater than or equal to the number of network outputs, in order to avoid hourglass structures, whose performance tends to be poor.

The evolutionary process adopts the well known convention that a lower fitness means a better ANN, mapping the objective function into an error minimization problem. Therefore, the fitness used for evaluating each individual in the population is proportional to the mean square error and to the cost of the considered network. The latter term induces a selective pressure favoring small networks.

4.1. ANN Encoding

Each individual in the population represents a multilayer perceptron (MLP). MLPs are feedforward neural networks with a layer of input neurons, a layer of one or more output neurons and zero or more 'hidden' (i.e., internal) layers of neurons in between; neurons in a layer can take inputs from the previous layer only.

A MLP is encoded in a structure in which basic information is maintained as illustrated in Table 1. The genotype encodes the number of hidden layers and the number of neurons for each hidden layer. We call this the *topology vector*. The input and output layers are identical for all the neural networks for a given task, but the size of the output layer depends on the number of the output classes for each benchmark dataset. Indeed, if the benchmark has only two target classes in the dataset, the network output will have only one neuron, defining what we call a *monoclass* problem; otherwise, the size of the output layer will have the same number of the dataset target classes, defining a *multiclass* problem.

The number of hidden nodes in the ith hidden layer corresponds to the number specified in the ith element in the topology vector (see Figure 1, the vector above the graph); furthermore, the chromosome of an individual encodes the connection weights of the corresponding MLP (see Figure 1, the reported graph). All nodes in each layer are connected to all nodes in the next layer as specified by the corresponding weight matrix, $\mathbf{W}$, defined for each pair of layers. Also, for each layer, the bias vector b is defined, which contains, in each element, the bias value of the corresponding node in the neural network.

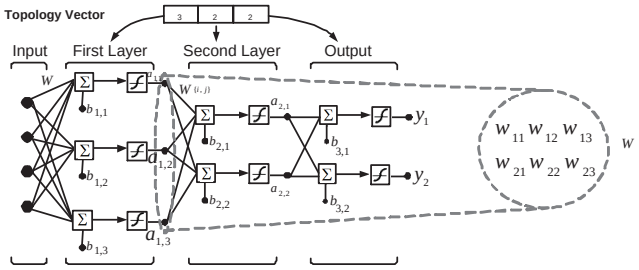

Figure 1: Representation of the ANN. The Topology vector reports the number of hidden nodes for each hidden layer and for the output layer. The graph explains all the connections among neural nodes and the node's biases. The matrix W shows the details of weights between two network's layers.

Initial weights and biases are extracted from a normal distribution, where the weights are always bounded in the interval [-5.0, 5.0]. Like in evolution strategies [40], for each connection weight and bias encoded in the genotype, an associated variance is also encoded, which determines the probability distribution of mutations and is used to self-adapt the mutation step.

The network topology is affected by the genetic operators during evolution, in order to perform both incremental (adding hidden neurons or hidden layers) and decremental (pruning hidden neurons or hidden layers) learning.

4.2. Evolutionary Process

In the evolutionary process, the genetic operators are applied to each network until the termination condition is satisfied, i.e. until the maximum number of generations is reached or no further

Table 1: Individual Representation.

Element	Description
topology	String of integer values that represent the number of neurons in each layer.
$\mathbf{W}^{(0)}$	Weights matrix of the input layer neurons of the network.
$\mathbf{Var}^{(0)}$	Variances for every element of $\mathbf{W}^{(0)}$.
$\mathbf{W}^{(i)}$	Weights matrix for the ith layer, $i = 1, \ldots, l$.
$\mathbf{Var}^{(i)}$	Variances every element of $\mathbf{W}^{(i)}$, $i = 1, \ldots, l$.
b_{ij}	Bias of the jth neuron in the ith layer.
$\mathrm{Var}(b_{ij})$	Variance of the bias of the jth neuron in the ith layer.

improvement of the fitness function can be obtained. In each new generation, a new population of size n has to be created, and the first half of such new population corresponds to the best parents that have been selected by the truncation operator, while the second part of the new population is filled with offspring of the previously selected parents. A child individual is generated by applying the crossover operator to two individuals, selected from the best half of the population (parents), if their local similarity condition is satisfied. Otherwise, the child corresponds to a randomly chosen copy of either of the two selected parents.

The crossover is applied with a probability parameter p_{cross}, defined by the user together with all the other genetic parameters, and maintained unchanged during the entire evolutionary process. A significant aspect related to the crossover probability considered in this work is that it refers to a 'desired' probability, a genetic parameter set at the beginning of the evolutionary process indicating the probability at which the crossover should be applied. However, the 'actual' crossover probability during the execution of the algorithm is less than or equal to the desired one, because the application of the crossover operator is conditional on a sort of 'compatibility' of the individuals involved. More details about this aspect are given in Section 5.

Elitism allows the survival of the best individual unchanged into the next generation. Then, the algorithm mutates the weights and the topology of the offspring, trains the resulting networks, calculates the fitness on the test set, and finally saves the best individual and statistics about the entire evolutionary process. Although the joint application of truncation selection and elitism could seem to exert a strong selection pressure, which could produce too fast a convergence of the solutions, all the experiments carried out with the `SimBa` crossover outperform the other approaches without showing any premature convergence of the results.

The general framework of the evolutionary process can be described by the following pseudo-code. Individuals in a population compete and communicate with other individuals through genetic operators applied with independent probabilities, until the termination condition is met.

1. Initialize the population by generating new random individuals.
2. Create for each genotype the corresponding MLP, and calculate its cost and its fitness values.
3. Save the best individual as the best-so-far individual.
4. While not termination condition do:
   (a) Apply the genetic operators to each network.
   (b) Decode each new genotype into the corresponding network.
   (c) Compute the fitness value for each network.
   (d) Save statistics.

The application of the genetic operators to each network is described by the following pseudo-code:

1. Select from the population (of size n) $\lfloor n/2 \rfloor$ individuals by truncation and create a new population of size n with copies of the selected individuals.
2. For all individuals in the population:
   (a) Randomly choose two individuals as possible parents.
   (b) If their local similarity is satisfied
      i. then generate the offspring by applying crossover according to the crossover probability.
      ii. else generate the offspring by randomly choosing either of the two parents.
   (c) Mutate the weights and the topology of the offspring according to the mutation probabilities.
   (d) Train the resulting network using the training set.
   (e) Calculate the fitness f on the test set.
   (f) Save the individual with lowest f as the best-so-far individual if the f of the previously saved best-so-far individual is higher (worse).
3. Save statistics.

For each generation of the population, all the information of the best individual is saved.

Table 2 lists all the parameters of the algorithm; the values they take up, reported in the third column, have been experimentally found as those that provide the most satisfactory results.

Table 2: Parameters of the Algorithm.

Symbol	Meaning	Default Value
n	Population size	60
p^{+}_{layer}	Probability of inserting a hidden layer	[0.05,0.15,0.30,0.45]
p^{-}_{layer}	Probability of deleting a hidden layer	[0.05,0.15,0.30,0.45]
p^{+}_{neuron}	Probability of inserting a neuron in a hidden layer	[0.05,0.15,0.30,0.45]
p_{cross}	'Desired' probability to apply crossover	[0.2,0.4,0.6,0.8,1.0]
δ	Crossover similarity cut-off value	0.1
N_{in}	Number of network inputs	$*$)
N_{out}	Number of network outputs	$*$)
α	Cost of a neuron	2
β	Cost of a synapse	4
λ	Desired trade-off between network cost and accuracy	0.2
k	Constant for scaling cost and MSE in the same range	10^{-6}

$*$) Benchmark dataset dependent.

4.2.1. Selection

Truncation selection, the selection method implemented in this work, is taken from the breeder genetic algorithm [41], and differs from natural probabilistic selection in that evolution only considers the individuals that best adapt to the environment. Truncation selection is not a novel solution and previous work considered such selection in order to prevent the population from remaining too

static and perhaps not evolving at all [42]. It is a very simple technique which produces satisfactory solutions in conjunction with other strategies, like elitism, which allows the best individual to survive unchanged into the next generation and solutions to monotonically get better over time.

4.2.2. Mutation

The main function of this operator is to introduce new genetic material and to maintain diversity in the population. Generally, the purpose of mutation is to simulate the effects of transcription errors that can occur with a very low probability, the mutation rate, when a chromosome is replicated. The evolutionary process applies two kinds of neural network perturbations: weights mutation and topology mutation.

Weights mutation perturbs the weights of the neurons before performing any structural mutation and applying BP. This kind of mutation uses a Gaussian distribution with zero mean and variance given by matrix $\mathbf{Var}^{(i)}$ for each network weight $\mathbf{W}^{(i)}$, as illustrated in Table 1. This solution is similar to the approach implemented by *evolution strategies* [43], algorithms in which the strategy parameters are proposed for self-adapting the mutation concurrently with the evolutionary search. The main idea behind these strategies is to allow a control parameter, like mutation variance, to self-adapt rather than changing its value according to some deterministic algorithm. Evolution strategies perform very well in numerical domains, and are well-suited to (real) function optimization. This kind of mutation offers a simplified method for self-adapting each single value of the Variance matrix $\mathbf{Var}_j^{(i)}$, whose values are defined as log-normal perturbations of their parent parameter values.

Topology mutation is defined with four types of mutation by considering neurons and layer addition and elimination. It is implemented after weight mutation because a perturbation of weight values changes the behavior of the network with respect to the activation functions. The addition and the elimination of a layer and the insertion of a neuron are applied with independent probabilities, corresponding respectively to the three algorithm parameters p^{+}_{layer}, p^{-}_{layer}, and p^{+}_{neuron}, while a neuron is eliminated only when its contribution becomes negligible with respect to the overall behavior of the network [24]. The parameters used in such kind of mutation are set at the beginning and maintained unchanged during the entire evolutionary process.

All the topology mutation operators are aimed at minimizing their impact on the behavior of the network; in other words, they are designed to be as little disruptive, and as much neutral, as possible, preserving the behavioral link between the parent and the offspring better than by adding random nodes or layers.

4.2.3. Fitness Function

Although it is customary in EAs to assume that better individuals have higher fitness, the convention that a lower fitness means a better ANN is adopted in this work. This maps directly to the objective function of an error- and cost- minimization problem, which is the natural formulation of most problems ANNs can solve.

In this work, the fitness function used depends on the number of the output classes of the considered benchmark problem. In particular, if the outputs are divided in more than two classes, we need to consider the benchmark as a multi-class problem. In such case, the fitness of an individual is defined as a function of the confusion matrix M obtained by that individual,

$$f_{multiclass}(M) = N_{\text{outputs}} - \text{Trace}(M), \tag{1}$$

where N_{outputs} is the number of output neurons (i.e., the number of senses) and $\text{Trace}(M)$ is the sum of the diagonal elements of the row-wise normalized confusion matrix, which represent the

conditional probabilities of the predicted outputs given the actual ones.

Generally, a confusion matrix is defined as a visualization tool typically used in supervised learning, and contains information about actual and predicted classifications performed by a classifier.

$$M = \begin{pmatrix} \Pr[c_1|c_1] & \cdots & \Pr[c_N|c_1] \\ \vdots & \ddots & \vdots \\ \Pr[c_1|c_N] & \cdots & \Pr[c_N|c_N] \end{pmatrix}, \tag{2}$$

where $c_1, \ldots, c_N$ are the classes, and $\Pr[c_i|c_j]$ is the probability that an instance is predicted to be in class c_i given that its actual class is c_j.

Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. The performance of a classifier is commonly evaluated using the data in the matrix, and the ideal results are obtained when all the elements of the confusion matrix belong to the diagonal of that matrix. Normalizing the rows of the confusion matrix enables us to solve many of the problems related to training a classifier with an unbalanced dataset, like in this application. In the other case, if the benchmark has only two outputs, like for example in the Credit Card or Heart Disease problems, we can consider it as a mono-class output problem.

Then, the fitness of an individual depends both on its accuracy (i.e., its mean square error mse) and on its cost, becoming therefore proportional to the value of the mse and to the cost of the considered network. It is defined by Equation 3:

$$f_{monoclass} = \lambda kc + (1 - \lambda)\text{mse}, \tag{3}$$

where $\lambda \in [0, 1]$ is a parameter which specifies the desired trade-off between network cost and accuracy, k is a constant for scaling the cost and the mse of the network to a comparable scale, and c is the overall cost of the considered network, that is proportional to the number of hidden neurons and synapses of the network topology. An interesting aspect could regard the study of multi-objective criteria in the fitness function definition, in order to more deeply study the behaviors changes when the objective function includes orthogonal issues like complexity. However, at this stage, we only consider parameter values that are the best experimentally found, as indicated in Table 2.

The mse depends on the *Activation Function*, that calculates all the output values for each single layer of the neural network. In this work we use the *Sigmoid Transfer Function*. The rationale behind introducing a cost term in the objective function is that we seek networks that use a reasonable amount of resources (neurons and synapses), which makes sense in particular when a hardware implementation is envisaged or the obtained ANNs have to be fast to compute.

5. The `SimBa` Crossover Operator

We have already discussed some of the most relevant critical issues related to the use of a crossover operator in a neuro-evolutionary context. In particular, two main drawbacks arise from the permutation problem, and correspond, respectively, to the structural incompatibility between two individuals, due to the different network topologies (genotypes with different lengths), and to the parametric incompatibility, which is related to the difference of the weight values associated to the network synapses [20]. The `SimBa` crossover operator proposed in this work aims at overcoming such drawbacks. The recombination is carried out in four phases, as follows:

Phase 1 (Figure 2): two individuals are selected from the population and the algorithm looks for a 'local similarity' between them, in order to define the two parents of the crossover operator. We refer to 'local similarity' as a situation in which, in both individuals, there are two consecutive layers (i and $i+1$) with the same number of neurons. This is a necessary condition for the application of our crossover operator because, this way, we want to overcome the problem related to the structure incompatibility between the individuals. If this condition is satisfied, layer $i+1$ is selected for the application of the crossover operator, otherwise no crossover is applied and the offspring will correspond to a randomly chosen copy of either of the two parents. The condition for the application of the operator is checked for all hidden layers. Therefore, two compatible individuals may have more than one crossover point.

Phase 2 (Figure 3): for each neuron of the selected layer $i+1$ of the two parents, the algorithm computes the corresponding contribution (i.e., the output obtained by evaluating the neural network, up to that neuron, over the training dataset), which strongly depends on its input connections. Then, the neurons of layer $i+1$ in each parent are ranked according to their contribution. Figure 3 shows an example of the contributions of the neurons, together with the calculated rank.

Phase 3 (Figure 4): we exploit the rank computed in the previous step to create the associations between the neurons of the two individuals. For each instance of the training set we compare the output of each neuron of layer $i+1$ in Individual 1 with the output of the each neuron of layer $i+1$ of Individual 2 and we compute the overall difference between the neurons (Figure 4a). The difference between two neurons is computed by Equation 4:

$$\text{Diff} = \frac{\sum_1^T |O_{I_1} - O_{I_2}|}{T} \tag{4}$$

where T is the number of training instances, and O_{I_1}, O_{I_2} are the outputs of the neurons respectively of Individual 1 and Individual 2. Such difference ranges in the $[0, 1]$ interval. Then, we choose a cut-off threshold δ that will be used in the next phase for the swap. Starting from the neuron of Individual 1 that has the highest contribution, we associate each neuron of Individual 1 with the neuron of Individual 2 that has the most similar behavior, i.e. the lowest output difference (Figure 4b). When an association is created, the associated neuron of Individual 2 becomes unavailable for subsequent associations (in the proposed example, neuron 15 is associated with neuron 26; therefore, neuron 26 could not be associated to any other neuron of Individual 1). Finally, the neurons of layer $i+1$ in Individual 2 are reordered according to the similarity associations with the neurons of layer $i+1$ in Individual 1 (Figure 4c).

Phase 4: the last phase of the algorithm generates the offspring. All the neuron associations linked with an output difference higher than the cut-off value δ are discarded. Such cut-off value δ can be seen as the 'local-similarity' threshold. In this work it has been experimentally defined at the beginning equal to 0.1 and maintained unchanged during the entire evolutionary process. The dashed line in Figure 5 separates the associations with an output difference higher than the δ value from the other ones. Only the neurons above such similarity threshold are eligible for the swap, while the others will remain unchanged. A cut-point is then randomly selected among such eligible neurons in the layer (Figure 5), then the algorithm swaps the weights of the neurons that are above the cut-point, maintaining unchanged all the others (Figure 6).

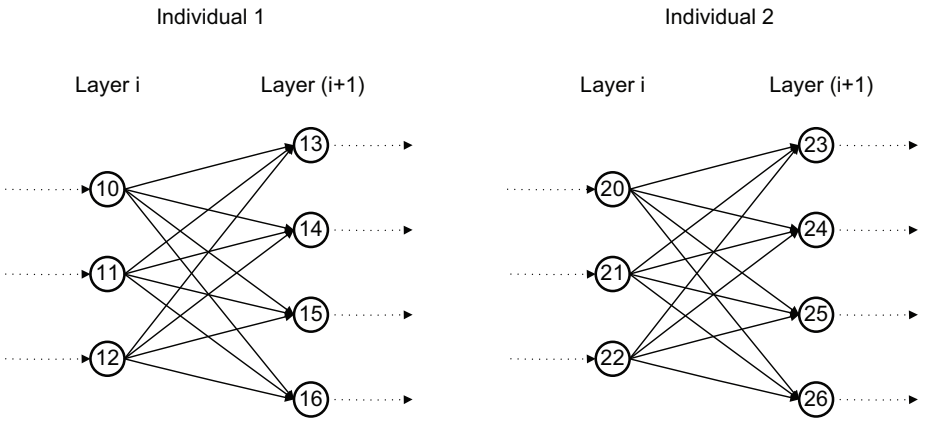

Figure 2: Phase 1: two individuals, Individual 1 and Individual 2, are chosen since they have two hidden layers, i and $i+1$, with the same number of neurons.

Neuron	Contr.
15	0.95
13	0.84
16	0.52
14	0.37
23	0.87
26	0.69
24	0.55
25	0.40

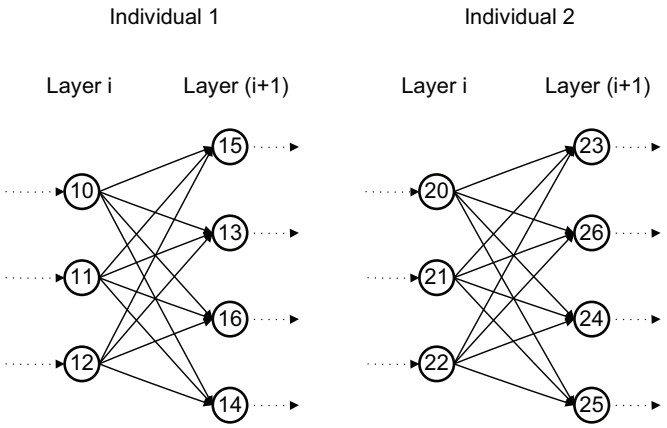

Figure 3: Phase 2: the contribution of each neuron of layer $i+1$ of the two parents is computed, then the neurons of the $(i+1)$th layer of the two parents are reordered according to their contribution. An example of the neuron contributions is shown in the table.

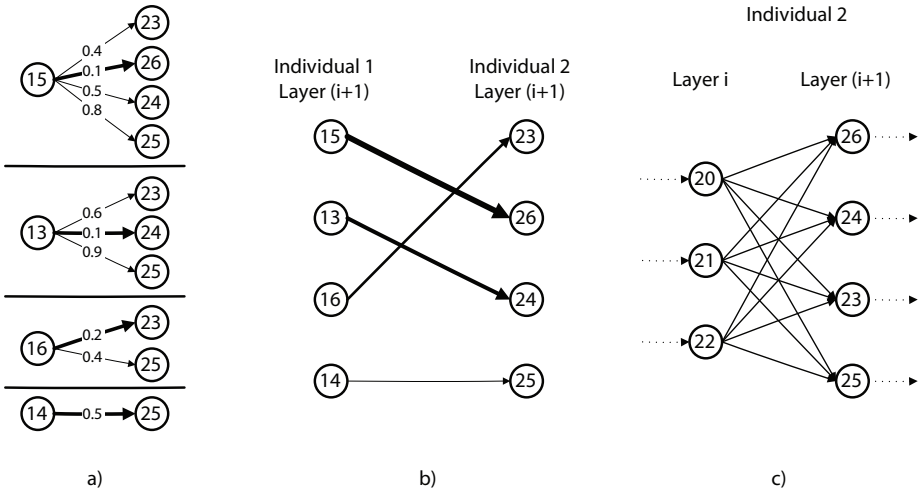

Figure 4: Phase 3: (a): the difference between the neuron outputs is obtained according to the Equation 4. (b): the output of each neuron of layer $i+1$ on Individual 1 is compared to the output of the each neuron of layer $i+1$ of Individual 2. (c): the neurons of layer $i+1$ in Individual 2 are reordered according to the similarity associations with the neurons of layer $i+1$ in Individual 1.

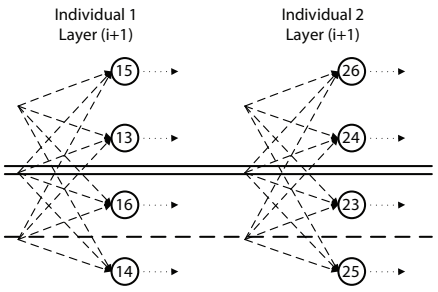

Figure 5: Phase 4a: a cut-point (the double continuous line) is randomly selected among all the neurons that have an output difference lower than the cut-off value δ (the dashed line), then the neurons above the cut-point are swapped.

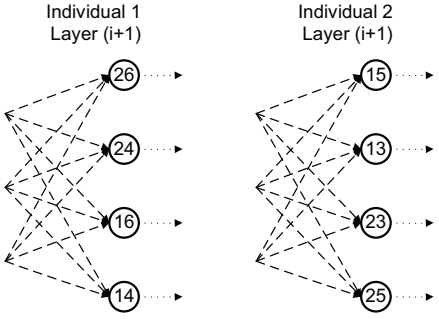

Figure 6: Phase 4b: the offspring generated by the crossover operator.

This use of similarity may raise the suspicion that, in fact, the proposed crossover operator does not perform any significant changes on the networks, due to the hypothesis that the recombination of similar neurons brings about a scenario where the results of recombination are very similar to their parents. To clear this suspicion, we now present an example that could be considered a limit case, in which we show what happens if we assume to recombine two neurons that are 100% similar, i.e. with an output difference equal to zero, on a given instance of the considered dataset.

We assume having two individuals I_1 and I_2, a selected layer $i+1$ on which we perform the crossover, and two neurons N_1 and N_2 coming respectively from I_1 and I_2, that are associated and, therefore, that will be recombined. We also assume that, for both I_1 and I_2, the previous layer i has three neurons in order to satisfy the 'local similarity' constraint. By starting from the following scenario, the weight vectors of the two neurons are defined as follows:

$$W_{I_1} = [0.4, 0.5, 0.7] \tag{5}$$
$$W_{I_2} = [0.7, 0.4, 0.5] \tag{6}$$

and the outputs of the neurons of layer i correspond to:

$$O_{I_1}^{L_i} = [0.6, 0.8, 0.2] \tag{7}$$
$$O_{I_2}^{L_i} = [0.2, 0.6, 0.8]. \tag{8}$$

When the output of each neuron of layer $i+1$ is computed, we will notice that, on the considered instance, the neurons are 100% similar (i.e. with an output difference equal to zero):

$$C(N_1) = (0.4 \cdot 0.6) + (0.5 \cdot 0.8) + (0.7 \cdot 0.2) = 0.78 \tag{9}$$
$$C(N_2) = (0.7 \cdot 0.2) + (0.4 \cdot 0.6) + (0.5 \cdot 0.8) = 0.78. \tag{10}$$

Then, the two neurons are recombined, by swapping the corresponding weight vectors:

$$W_{I_1} = [0.7, 0.4, 0.5] \tag{11}$$
$$W_{I_2} = [0.4, 0.5, 0.7]. \tag{12}$$

After such recombination, we can observe how the new outputs of the two neurons differ from the initial values:

$$C(N_1) = (0.7 \cdot 0.6) + (0.4 \cdot 0.8) + (0.5 \cdot 0.2) = 0.84 \tag{13}$$
$$C(N_2) = (0.4 \cdot 0.2) + (0.5 \cdot 0.6) + (0.7 \cdot 0.8) = 0.94. \tag{14}$$

Even if in this particular example only one scenario of the recombination process is presented, generally the use of 'local similarity' does not lead to similar neurons both before and after the application of the crossover operator, avoiding a premature convergence of the solutions.

6. Experiments and Results

The approach described above has been applied to six well-known benchmark classification problems, namely Credit Card, Glass, Heart, Ionosphere, Iris, and, finally, the Pima Indian Diabetes problem. The corresponding datasets, obtained from the UCI Machine Learning Repository are partitioned into three non-overlapping sets, respectively, a training set (50% of the data), used to

train the network; a validation set (25% of the data), used to stop the training and avoid overfitting, and a test set (25% of the data), used to test the generalization capabilities of a network. Such a dataset partition has been performed according to the guidelines presented in [44].

The input attributes of all datasets have been rescaled, before being fed as inputs to the population of ANNs, through a Gaussian distribution with zero mean and standard deviation equal to one.

All the experiments have been carried out by considering different settings for both the topology mutation parameters p^{+}_{layer},p^{-}_{layer} and p^{+}_{neuron} and for the crossover p_{cross}, the 'desired' probability, (see Section 4.2). The values these parameters take up are chosen in the corresponding sets defined in Table 2 in order to determine the setting that yields the best performance. For each different configuration, we performed 20 runs, with 40 generations and 60 individuals set for each run. The number of epochs used to train the neural network implemented in each individual has been set to 250 when the evolutionary algorithm has been applied, while in the other, traditional cases, the number of epochs has been set to 1,000,000. Such a high number of epochs has been used in these cases in order to better test and compare non-evolutionary approaches, i.e. Backpropagation (BP) and Conjugate Gradient algorithms, but also to demonstrate the capabilities and the performance of the evolutionary approach implemented in this work, even though its training phase is carried out with a reduced number of epochs.

The synthesis of the results is shown in Tables 3 to 8.

In order to show the performance and the advantages of applying such novel evolutionary method, for all the benchmark problems above reported, we compared it with five different baseline approaches, by considering the same number of runs and individuals used to define a population in our evolutionary approach:

- Simple ANN with BP: the classifiers are encoded with a population of ANNs, trained with 1,000,000 epochs. The networks are then evaluated over, respectively, the validation and the test sets, through the application of the mean square error.
- Simple ANN with Conjugate Gradient: the classifiers are encoded with a population of ANNs, trained with the Conjugate Gradient method [45] over 1,000,000 epochs. Also in this case, the networks are then evaluated over, respectively, the validation and the test sets, through the application of the mean square error.
- NeuroEvolution of Augmenting Topologies (NEAT) approach [2]: an evolutionary approach applied to neural network design that: (1) uses a crossover on different topologies, (2) protects structural innovation by using speciation, and (3) applies an incrementally growing from minimal network structures.
- Evolved ANN without crossover: the population of ANNs are evolved through the joint optimization of architecture and connection weights reported in this work, but in this case no crossover is implemented. The number of epochs corresponds to 250.
- Evolved ANN with naive crossover: the population of ANNs are evolved through the joint optimization of architecture and connection weights, reported in this work, and a naive crossover operator. Such a crossover randomly defines a cut-point and swaps the neurons in the corresponding hidden layers. The number of epochs corresponds to 250.

The choice of all such baseline approaches enables us to better evaluate our approach by extending the comparison both to:

- traditional approaches: simple ANN with BP, which is one of the most used examples of supervised learning algorithms, and also applied as classifiers, and simple ANN with CG, a higher-order optimization method regarded as more powerful than BP.
- evolutionary approaches: NEAT, which is one of the most popular state-of-the-art methods for ANN evolution, implementing an approach based on the augmenting topology, an evolutionary approach without crossover, and, finally, a naive crossover approach that recombines individuals by randomly swapping the hidden layers. The latter two algorithms aim to study, respectively, the influence of the application of the mutation operator in an evolutionary process, and the influence of the addition of a naive crossover that does not try to overcome the above-mentioned permutation problem.

After a brief analysis of the results, we can observe that the application of the `SimBa` crossover leads to an improvement of the results. In the following section we will discuss the impact of the proposed operator and the significance of the obtained results.

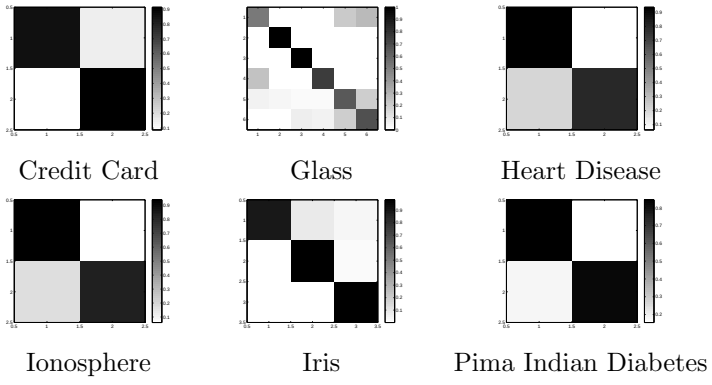

Figure 7: Confusion matrices of the benchmark problems analyzed by the SimBa evolutionary approach.

Moreover, in order to provide a more detailed, critical discussion of the results, it will be convenient to also consider the graphical representation of the corresponding (row-wise normalized) confusion matrices, defined as in Equation 2. All the confusion matrices for the considered benchmarks are shown in Figure 7. A visual analysis confirms the satisfactory performances obtained by the `SimBa` crossover approach, both for mono and multi-classes problems, as shown in Tables 3 to 8.

6.1. Discussion

As previously indicated, the application of the `SimBa` crossover has a significant impact over the network performances and basic statistical information such as the best and average accuracy and the standard deviation may be helpful in studying the effectiveness of such operator with respect to the baselines. Moreover, all the experimental results have been statistically verified by applying Student's t test in order to check their significance. Tables 3 to 8 show the results obtained from the experiments for all the considered benchmark problems.

From a first analysis, it is possible to see that the Simple ANN with BP always obtains the worst performance compared to the other approaches, and that the application of the naive crossover

always leads to worse results than both NEAT and the evolved ANN without crossover. The reasons of these poor performances may be found in the structure of the naive crossover. Indeed, in such operator, the recombination is carried out in a random way without considering the topology of the parent networks, thus producing detrimental effects on the quality of the generated offspring. Moreover, from a preliminary analysis carried out among the evolutionary approaches that make the application of the crossover operator an important and useful aspect, NEAT allows to obtain, after the `SimBa` crossover, the best classification results, albeit at the cost of a slight increase in the network size.

A different behavior is exhibited by the `SimBa` crossover, and the obtained results prove that the local similarity is an effective strategy, which is able to generate offspring by identifying regions over the network topologies that may be recombined without loss of information or disruption, while at the same time preventing the premature convergence of the overall algorithm. Moreover, as stated in Section 5, the recombination by selecting 'similar' neurons cannot be compared to a mutation operator, since the differences between the parents and the offspring cannot be reproduced only by mutating the topologies and the weights, even if applied in conjunction.

Another interesting aspect is that the application of different crossover probabilities does not lead to significantly different results. Indeed, by observing the average accuracies in Tables 3 to 8, we can notice that different crossover probabilities usually lead to quite close accuracy values, and the differences are not statistically significant. On four datasets (Credit Card, Heart, Ionosphere, and Pima) the accuracies are obtained with high probability values (0.8 and 1.0), while on the other two datasets (Glass and Iris) the best accuracies are obtained with lower probability values (0.2 and 0.4), even if very similar results have also been obtained with the higher probability values. Such aspect also shows the effectiveness and the robustness of the presented neuroevolutionary approach with the `SimBa` crossover with respect to changing parameters.

By observing these results, we cannot infer which is the ideal usage probability of the operator; however, we have verified its effectiveness with high probability values. This fact coincides with the notion that, as stated in [46], evolutionary processes generally apply crossover operators with high probability values. Therefore, the operator is suitable for evolutionary processes that use standard parameter settings.

This fact demonstrates that, despite applying the crossover few times during the evolutionary process, the approach based on the `SimBa` crossover is able to perform effective recombinations of individuals in order to increase the exploration range of the solution space.

It should be noticed that this novel crossover operator does not negatively impact the overall computational cost of the evolutionary process, because it exploits information previously calculated and saved during the learning phase, and it only implements comparison of such network information in order to define the right associations among neurons and then swap the corresponding connections. Nor the size of the evolved networks is affected by the application of `SimBa`.

All such considerations are validated by the network sizes reported in Tables 3 to 8. The first important comparison regards the differences that arise from the application of traditional or evolutionary approaches. Indeed, it is possible to observe that, for all the benchmark datasets, the use of an evolutionary process allows to obtain networks that are dramatically smaller than those obtained without the use of evolution, at the same time providing more satisfactory accuracies.

Another interesting observation regards the application of the crossover operator in the neuroevolutionary approaches: as a natural consequence of the application of the augmenting topologies algorithm, the network sizes obtained by NEAT are the largest ones among the crossover-based approaches, while comparisons between `SimBa` and a naive crossover demonstrate that the `SimBa`

crossover has a negligible impact on the network sizes, with consequences over the computational cost as stated above.

Finally, a general observation is that, for all the datasets, the evolutionary process finds more compact networks, which also means lower computational cost and more robust solutions. We highlight that, when the number of classes increases, the average size of the networks increases as well and, even if this leads to a small increase of the computing time, this fact is justified by the necessity of preventing bottlenecks in the networks in order to maintain high accuracies.

Table 3: Summary of the results obtained with different crossover probabilities on the CREDIT CARD dataset.

p_{cross}	Best	Average	Std Dev	Avg Size of Best	Global Avg Size
Simple ANN with CG					
-	0.8242	0.7967	0.0287	6.4984	5.2874
Simple ANN with BP					
-	0.8089	0.7818	0.0212	6.5355	5.1654
Evolved ANN without Crossover					
0.0	0.8912	0.8608	0.0209	2.7049	2.3314
NEAT					
0.2	0.8859	0.8618	0.0201	5.0783	3.8702
0.4	0.8837	0.8571	0.0212	5.1819	3.9479
0.6	0.8814	0.855	0.0197	5.1329	3.9111
0.8	0.8951	0.8658	0.0191	4.449	3.746
1.0	0.8916	0.8624	0.0193	4.7581	3.7605
Evolved ANN with Naive Crossover					
0.2	0.8701	0.8407	0.0212	4.2779	3.0426
0.4	0.8706	0.8426	0.0201	4.1732	2.9657
0.6	0.8721	0.8429	0.0197	4.2274	3.006
0.8	0.8703	0.8453	0.0193	3.8525	2.8561
1.0	0.8688	0.8425	0.0191	3.5439	2.8408
Evolved ANN with SimBa Crossover					
0.2	0.9127	0.884	0.0214	4.3092	3.0473
0.4	0.9135	0.8856	0.0202	4.139	2.9803
0.6	0.9132	0.8851	0.0197	4.216	3.0136
0.8	0.9131	0.8874	0.0192	3.8503	2.8655
1.0	0.911	0.885	0.019	3.5946	2.8401

Table 9 shows the results of the statistical significance test performed by using Student's t distribution. We can observe that, by comparing our results with the the ANN without evolution (trained with Backpropagation or Conjugate Gradient) and the naive crossover, the significance is always greater than 95%. Also by considering the evolved ANN without the crossover operator the significance reaches satisfactory values, since it is greater than 90%. Only two exceptions are recorded for the Ionosphere and Glass datasets, but they refer to a minority in both cases. Moreover, by observing the result of the significance test performed with respect to NEAT, we can notice that, in most cases, the significance of the improvements is greater than 95%.

We have also compared the best results obtained by the proposed approach with the ones obtained by the most recent state-of-the-art algorithms based on the evolution of artificial neural networks. In Table 10 we show the results of the comparisons. In some cases, in the relevant literature, the approach we compare to was not tested on the same datasets as those we used: when that is the case, no results are reported. By looking at the results, we can observe that, in 6 cases out of 10, the proposed approach outperforms the others, and in one case (i.e., the GLASS average performance) the proposed approach is very close to the best (0.6588 vs. 0.6623).

Table 4: Summary of the results obtained with different crossover probabilities on the GLASS dataset.

p_{cross}	Best	Average	Std Dev	Avg Size of Best	Global Avg Size
Simple ANN with CG					
-	0.7204	0.5610	0.0208	15.1843	12.8976
Simple ANN with BP					
-	0.7064	0.5454	0.0186	15.8363	13.2843
Evolved ANN without Crossover					
0.0	0.7864	0.6274	0.0849	6.6996	6.4402
NEAT					
0.2	0.7788	0.6297	0.0819	12.061	9.1694
0.4	0.7761	0.6297	0.0809	11.8702	9.3806
0.6	0.7775	0.6306	0.0855	12.0256	9.0669
0.8	0.771	0.622	0.086	11.6459	8.7706
1.0	0.7816	0.6283	0.0856	12.1678	9.1268
Evolved ANN with Naive Crossover					
0.2	0.7561	0.6084	0.0809	10.9658	8.4763
0.4	0.758	0.6103	0.0819	11.1558	8.2645
0.6	0.7582	0.6103	0.0855	11.1205	8.1612
0.8	0.7616	0.6078	0.0856	11.2633	8.2219
1.0	0.7515	0.6017	0.086	10.7401	7.8657
Evolved ANN with SimBa Crossover					
0.2	0.8073	0.6572	0.0817	11.0571	8.493
0.4	0.8075	0.6588	0.0824	11.0607	8.3092
0.6	0.8058	0.6572	0.0856	11.0844	8.1803
0.8	0.8108	0.6571	0.0852	11.2669	8.2482
1.0	0.8007	0.6503	0.0858	10.8769	7.8578

Table 5: Summary of the results obtained with different crossover probabilities on the HEART dataset.

p_{cross}	Best	Average	Std Dev	Avg Size of Best	Global Avg Size
Simple ANN with CG					
-	0.8271	0.7847	0.0187	6.9842	4.9981
Simple ANN with BP					
-	0.8177	0.7765	0.0211	6.6837	5.2964
Evolved ANN without Crossover					
0.0	0.8787	0.8333	0.026	2.8692	2.4154
NEAT					
0.2	0.8708	0.8326	0.0235	4.9729	3.8217
0.4	0.8751	0.8317	0.025	4.9085	3.9237
0.6	0.8718	0.8338	0.0225	4.8029	3.7528
0.8	0.8748	0.8373	0.0216	4.5183	3.6481
1.0	0.8691	0.8347	0.0219	4.7188	3.7961
Evolved ANN with Naive Crossover					
0.2	0.8548	0.8115	0.025	4.0035	3.0185
0.4	0.8507	0.8122	0.0235	4.0683	2.9158
0.6	0.8519	0.8136	0.0225	3.8982	2.8469
0.8	0.8493	0.8146	0.0219	3.8143	2.8902
1.0	0.855	0.8175	0.0216	3.6141	2.7424
Evolved ANN with SimBa Crossover					
0.2	0.9031	0.8602	0.0253	4.0381	3.0246
0.4	0.8989	0.8603	0.0237	4.0335	2.9286
0.6	0.9003	0.8619	0.0226	3.8836	2.8525
0.8	0.8979	0.862	0.0219	3.8161	2.9
1.0	0.9028	0.8644	0.0216	3.66	2.7422

Table 6: Summary of the results obtained with different crossover probabilities on the IONOSPHERE dataset.

p_{cross}	Best	Average	Std Dev	Avg Size of Best	Global Avg Size
Simple ANN with CG					
-	0.8908	0.8463	0.0388	4.7254	4.1846
Simple ANN with BP					
-	0.8821	0.8318	0.0345	4.4862	4.7209
Evolved ANN without Crossover					
0.0	0.9179	0.8669	0.0314	3.0428	2.5065
NEAT					
0.2	0.9112	0.8644	0.0302	5.1036	4.0226
0.4	0.915	0.8594	0.032	4.8975	4.0888
0.6	0.9151	0.8681	0.0283	5.0717	3.9413
0.8	0.9124	0.872	0.0219	4.4838	3.8573
1.0	0.9156	0.8665	0.027	4.6099	3.9022
Evolved ANN with Naive Crossover					
0.2	0.8948	0.8407	0.032	3.9926	3.1832
0.4	0.8917	0.8444	0.0302	4.1983	3.1171
0.6	0.8948	0.8479	0.0283	4.1668	3.037
0.8	0.8951	0.8469	0.027	3.7061	2.9968
1.0	0.8919	0.8518	0.0219	3.5784	2.9526
Evolved ANN with SimBa Crossover					
0.2	0.9409	0.8864	0.0323	4.0267	3.1841
0.4	0.9351	0.8908	0.0305	4.1656	3.1317
0.6	0.9395	0.8937	0.0283	4.144	3.0487
0.8	0.9396	0.8944	0.027	3.7008	3.0084
1.0	0.9374	0.8992	0.0219	3.626	2.9544

Table 7: Summary of the results obtained with different crossover probabilities on the IRIS dataset.

p_{cross}	Best	Average	Std Dev	Avg Size of Best	Global Avg Size
Simple ANN with CG					
-	0.9075	0.9052	0.0008	7.2669	5.8792
Simple ANN with BP					
-	0.9021	0.9005	0.0014	7.9836	5.3337
Evolved ANN without Crossover					
0.0	0.9177	0.9164	0.001	4.0142	3.2817
NEAT					
0.2	0.9258	0.9092	0.0106	5.3528	6.6805
0.4	0.9277	0.9103	0.008	5.1939	6.6165
0.6	0.929	0.9084	0.0124	5.5431	6.5744
0.8	0.9413	0.9047	0.0175	6.1058	6.5019
1.0	0.9427	0.9071	0.0159	5.8038	6.5817
Evolved ANN with Naive Crossover					
0.2	0.9082	0.8906	0.008	4.2893	5.7118
0.4	0.9069	0.89	0.0106	4.4474	5.7752
0.6	0.9083	0.8886	0.0124	4.6383	5.6695
0.8	0.9229	0.8873	0.0159	4.8983	5.6771
1.0	0.9211	0.8849	0.0175	5.2001	5.5974
Evolved ANN with SimBa Crossover					
0.2	0.962	0.9445	0.0081	4.3279	5.7097
0.4	0.9593	0.9429	0.0107	4.4101	5.8027
0.6	0.9631	0.9424	0.0124	4.6317	5.6875
0.8	0.9761	0.9414	0.0158	4.8908	5.6971
1.0	0.9745	0.9391	0.0174	5.2641	5.5967

Table 8: Summary of the results obtained with different crossover probabilities on the PIMA Indian Diabetes dataset.

p_{cross}	Best	Average	Std Dev	Avg Size of Best	Global Avg Size
Simple ANN with CG					
-	0.7690	0.7400	0.0164	6.2762	5.3853
Simple ANN with BP					
-	0.7526	0.7308	0.0197	6.8673	5.1724
Evolved ANN without Crossover					
0.0	0.8233	0.8021	0.013	3.019	2.4476
NEAT					
0.2	0.83	0.7939	0.0218	4.9143	3.9062
0.4	0.8254	0.7955	0.0215	4.9287	3.9562
0.6	0.8256	0.7938	0.0224	4.7825	3.8055
0.8	0.8285	0.7919	0.019	3.6866	3.757
1.0	0.8291	0.7955	0.0221	4.4773	3.6988
Evolved ANN with Naive Crossover					
0.2	0.7909	0.7606	0.0215	4.0241	3.0515
0.4	0.7953	0.7586	0.0218	4.0105	3.0020
0.6	0.7917	0.7595	0.0224	3.8769	2.9007
0.8	0.7936	0.7605	0.0221	3.5726	2.7939
1.0	0.7924	0.7561	0.0190	2.7814	2.8521
Evolved ANN with SimBa Crossover					
0.2	0.8693	0.8381	0.0215	4.0107	3.0456
0.4	0.8708	0.8361	0.0216	3.9661	2.9740
0.6	0.8711	0.8363	0.0220	3.8357	2.8823
0.8	0.8713	0.8383	0.0219	3.5411	2.7588
1.0	0.8696	0.8343	0.0191	2.7786	2.8482

Table 9: Summary of the statistical significance test computed on the average results obtained on the Benchmark datasets.

Benchmark Problem	p_{cross}	Simple ANN with CG	Simple ANN with BP	Evolved ANN without Crossover	Naive Crossover	NEAT
Credit Card	0.2	100%	100%	94.32%	99.92%	93.53%
	0.4	100%	100%	96.03%	99.94%	98.04%
	0.6	100%	100%	95.74%	99.93%	98.79%
	0.8	100%	100%	97.40%	99.94%	93.86%
	1.0	100%	100%	95.87%	99.95%	94.90%
Glass	0.2	100%	100%	81.09%	96.73%	77.96%
	0.4	100%	100%	83.19%	96.55%	80.41%
	0.6	100%	100%	80.56%	95.51%	75.36%
	0.8	100%	100%	80.42%	96.46%	87.07%
	1.0	100%	100%	88.23%	96.14%	66.25%
Heart	0.2	100%	100%	92.17%	99.80%	93.60%
	0.4	100%	100%	92.70%	99.83%	94.44%
	0.6	100%	100%	94.50%	99.87%	94.94%
	0.8	100%	100%	94.74%	99.86%	92.18%
	1.0	100%	100%	96.42%	99.86%	96.42%
Ionosphere	0.2	98.75%	99.94%	81.48%	99.67%	86.56%
	0.4	99.45%	99.98%	89.80%	99.78%	96.52%
	0.6	99.71%	99.99%	93.67%	99.82%	93.14%
	0.8	99.77%	99.99%	94.56%	99.90%	91.47%
	1.0	99.94%	100%	98.07%	99.97%	98.60%
Iris	0.2	99.99%	100%	99.74%	100%	99.23%
	0.4	99.95%	99.99%	98.87%	99.97%	98.67%
	0.6	99.89%	99.97%	98.11%	99.94%	97.62%
	0.8	99.59%	99.86%	95.73%	99.80%	96.51%
	1.0	99.08%	99.64%	92.45%	99.70%	93.65%
Pima	0.2	100%	100%	95.82%	99.98%	97.38%
	0.4	100%	100%	94.63%	99.98%	96.06%
	0.6	100%	100%	94.61%	99.97%	96.59%
	0.8	100%	100%	95.83%	99.98%	98.31%
	1.0	100%	100%	94.21%	99.99%	95.61%

Table 10: Comparison between the proposed approach and the most recent state of the approaches based on the evolution of artificial neural networks.

Approach	CREDIT CARD	GLASS	HEART	IONOSPHERE	PIMA
Our Approach (Best)	**0.9135**	0.**8108**	0.9031	0.9409	**0.8713**
Our Approach (Avg)	**0.8874**	0.6588	**0.8644**	0.8992	**0.8383**
PSU [47] (Best)	0.8855	0.7014	0.8820	0.9399	0.8076
PSU [47] (Avg)	0.8738	**0.6623**	0.8593	**0.9211**	0.7889
q-Gaussian [48] (Best)	0.8824	0.7526	0.9099	N/A	N/A
q-Gaussian [48] (Avg)	0.8787	0.6532	0.8579	N/A	N/A
CMAC [49] (Best)	N/A	0.6242	N/A	N/A	0.7538
CMAC [49] (Avg)	N/A	0.6113	N/A	N/A	0.7452
ProductUnit [50] (Best)	N/A	N/A	**0.9630**	**1.0000**	0.8421
ProductUnit [50] (Avg)	N/A	N/A	0.8189	0.8963	0.7740

6.2. 'Desired' and 'Actual' crossover probability

In Section 4.2 we introduced the difference between the 'desired' probability and the 'actual' probability by saying that the 'actual' (i.e. the real) probability is always less than or equal to the desired one. This happens because the application of the crossover operator is conditional on the local-similarity between the individuals. In this section, we want to analyze in more detail this aspect of the proposed operator by observing which is the behavior of the 'actual' probability during the entire evolutionary process on the datasets used in our experiments.

Figures from 8 to 13 report the values of the 'actual' probability for each generation. Each point represents the average of the 'actual' probability of crossover operator computed over all the combinations of mutation probabilities used in all the experiments carried out in this work. By observing the shapes, we can identify three different phases that correspond to three different behaviors of the 'actual' probability: an initialization phase, corresponding to the first generation, in which the behavior of the 'actual' probability is conditioned by the random individual initialization; an exploration phase, corresponding to the core of the evolution in which the behavior of the 'actual' probability is influenced by the exploration of the solution space of the evolutionary algorithm; and a convergence phase, corresponding to the second half of the generations in which the behavior of the 'actual' probability follows the convergence of the evolutionary algorithm.

In the initialization phase we can observe two different behaviors of the 'actual' probability: values that are close to the 'desired' probability and values that are close to zero. This fact is caused by the random initialization of the population; indeed, a significant diversity of the generated individuals may lead to low values of the 'actual' probability. Such diversity in the topology initialization is also influenced by the number of the output classes for each considered benchmark. For example, for the datasets of Glass and Iris, with, respectively, 6 and 3 output classes, the starting 'actual' probability is close to zero for all the corresponding values of the 'desired' probability. At this stage, local similarity is hard to obtain. Instead, for the other four datasets, that have only two classes, corresponding to just one output neuron, the diversity of the population at the initialization is lower; therefore, it is easier for the crossover operator to find parents that are locally similar.

A similar scenario occurs in the exploration phase, where, again, similar topologies are difficult to find. Examples are given by Credit Card, Heart, Ionosphere, and Pima, for which a decrease of the 'actual' probability is reported. This occurs because, after the initialization phase, especially on datasets for which the generated individuals are quite similar, the algorithm explores different

network topologies in the solution space. Therefore, the crossover operator has more difficulties in finding candidate parents that are locally similar in order to generate new offspring.

Finally, when the algorithm starts the final convergence phase, after the exploration phase, the probability of finding parents that are locally similar increases, with a consequent decrease of the difference between 'actual' and 'desired' probabilities.

Table 11: Actual and desired values of the crossover probability for the considered benchmark problems.

p_{cross}	Desired Setting
All Benchmark	[0.2 0.4 0.6 0.8 1.0]
p_{cross}	Actual Setting
Iris	[0.28 0.49 0.68 0.80 0.89]
Glass	[0.12 0.26 0.40 0.56 0.76]
Ionosphere	[0.07 0.16 0.30 0.52 0.91]
Credit Card	[0.12 0.23 0.38 0.53 0.79]
Heart	[0.11 0.24 0.38 0.55 0.78]
Pima	[0.09 0.18 0.31 0.45 0.90]

Table 11 shows how the 'actual' crossover probability changes for each benchmark problem that we considered, according to a best setting reached during the evolutionary process. Such values can be compared to the 'desired' probability shown at the top of the table.

The actual values reported show that, except for the Iris setting, for all the other benchmark problems the actual values are always less than the desired ones.

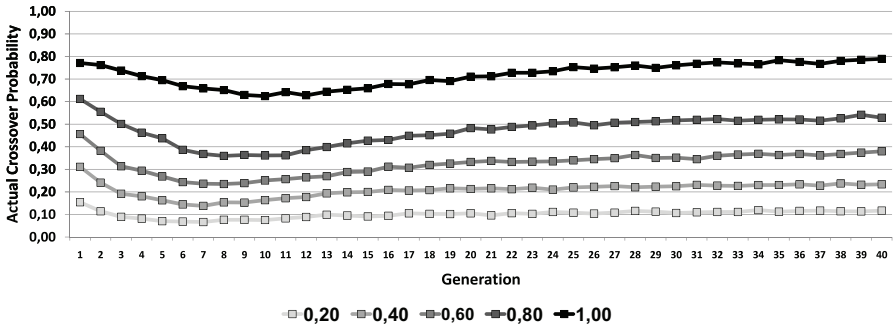

Figure 8: Trend of the actual crossover probability on the CREDIT CARD dataset.

7. Conclusion and Future Work

We have presented `SimBa`, a novel similarity-based crossover operator, applied in conjunction with weights and topology mutations, for the evolution of artificial neural networks. The crossover operator considers the similarity between the neuron outputs in order to choose which neurons may be swapped between the networks. The approach has been validated over six well-known benchmark problems from the UCI Repository. Moreover, according to the number of output classes presented by each benchmark, in this neuro-genetic approach two kinds of fitness function are defined, named respectively, mono and multi class function. All the experiments carried out by comparing different methodologies, both traditional and evolutionary, show that the application

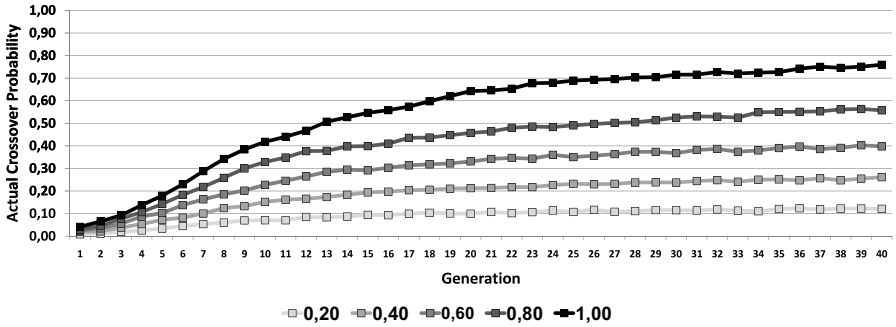

Figure 9: Trend of the actual crossover probability on the GLASS dataset.

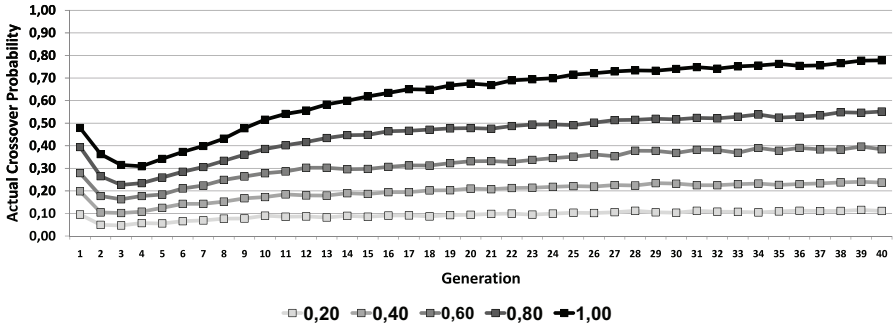

Figure 10: Trend of the actual crossover probability on the HEART dataset.

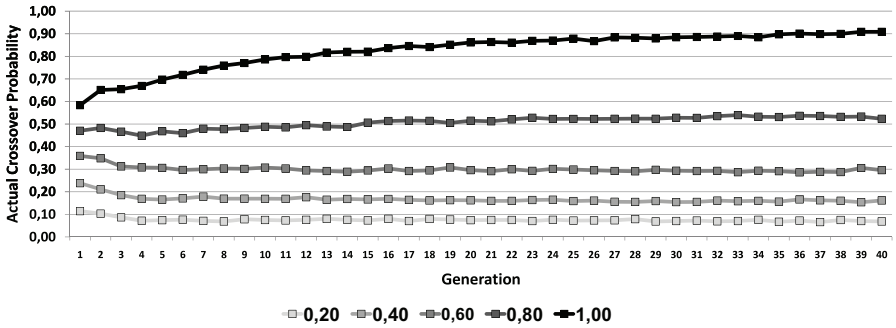

Figure 11: Trend of the actual crossover probability on the IONOSPHERE dataset.

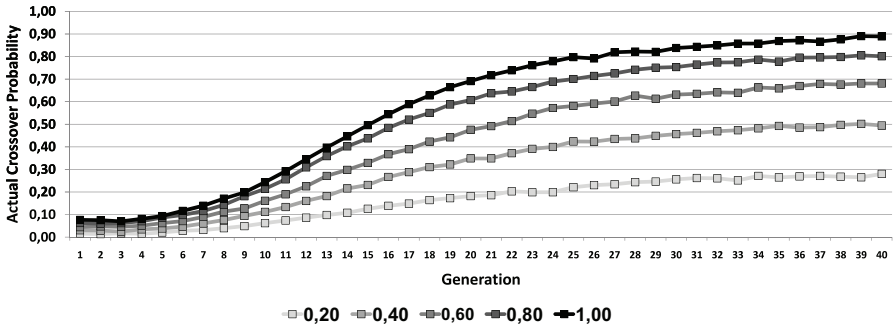

Figure 12: Trend of the actual crossover probability on the IRIS dataset.

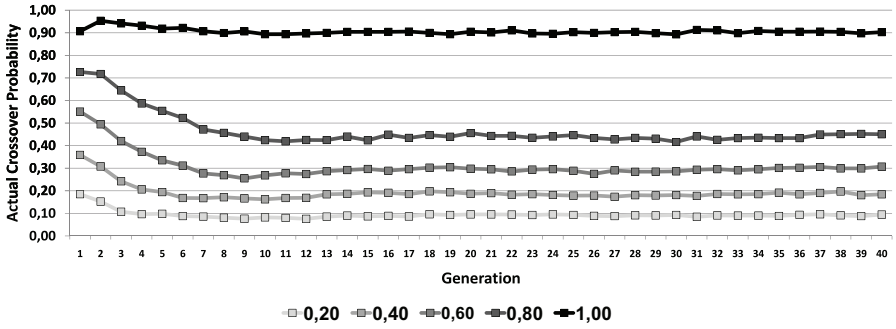

Figure 13: Trend of the actual crossover probability on the PIMA Indian Diabetes dataset.

of a similarity-based crossover operator to a well-tested neuro-genetic approach achieves promising results, by demonstrating the viability of such operator.

Two kinds of crossover probabilities, 'desired' and 'actual', are presented and discussed in the paper, and the experimental results show how the 'actual' probability is always less than or equal to the desired one. This happens because, in this work, the application of the crossover operator is conditional on the local-similarity between the individuals. The overall accuracy, however, confirms satisfactory performances with reduced computational costs, also thanks to the capability of the algorithm to evolve small network topologies.

An interesting aspect that could be investigated in the next future regards the use of multiple criteria instead of a single-objective fitness function, to better separate orthogonal issues like accuracy and complexity. An additional interesting aspect that could be analyzed is the influence of different similarity thresholds of the SimBa crossover operator over the implemented evolutionary approach.

References

[1] D. Fogel, The advantages of evolutionary computation, in: Proc. of the Int. Conf. on Biocomputing and Emergent Computation, BCEC'97, World Scientific, 1997, pp. 1–11.

[2] K. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evolutionary Computation 10 (2002) 99–127.

[3] P. Wang, T. Weise, R. Chiong, Novel evolutionary algorithms for supervised classification problems: an experimental study, Evolutionary Intelligence Special Issue (2011) 1–14.

[4] X. Yao, Y. Xu, Recent advances in evolutionary computation, International Journal on Computer Science and Technology 21 (2006) 1–18.

[5] X. Yao, Y. Liu, A new evolutionary system for evolving artificial neural networks, IEEE Transactions on Neural Networks 8 (1997) 694–713.

[6] D. Floreano, P. Durr, C. Mattiussi, Neuroevolution: from architectures to learning, Evolutionary Intelligence 10 (2008) 47–62.

[7] P. Hancock, Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification, in: Proc. of IEEE Int. Workshop on Combinations of Genetic Algorithms and Neural Networks, COGANN'92, IEEE Press, 1992, pp. 108–122.

[8] N. Radcliffe, Genetic set recombination and its application to neural network topology optimization, Neural Computing and Applications 1 (1993) 67–90.

[9] N. Garcia-Pedrajas, C. Hervas-Martinez, J. Munoz-Perez, COVNET: A cooperative coevolutionary model for evolving artificial neural networks, IEEE Transactions on Neural Networks 14 (2003) 575–596.

[10] A. Azzini, M. Dragoni, A. Tettamanzi, A novel similarity-based crossover for artificial neural network evolution., in: Proceedings of the XI International Conference on Parallel Problem Solving from Nature, PPSN'10 - Lecture Notes in Computer Science, volume 6238, Springer, 2010, pp. 344–353.

[11] A. Azzini, A. G. B. Tettamanzi, M. Dragoni, Simba-2: Improving a novel similarity-based crossover for the evolution of artificial neural networks, in: S. Ventura, A. Abraham, K. Cios, C. Romero, F. Marcelloni, J. M. Benitez, E. Gibaja (Eds.), Proceedings of the 2011 11th International Conference on Intelligent Systems Design and Applications (ISDA 2011), IEEE, 2011, pp. 374–379.

[12] A. Azzini, A. Tettamanzi, A new genetic approach for neural network design, in: A. Abraham, C. Grosan, W. Pedrycz (Eds.), Engineering Evolutionary Intelligent Systems. Studies in Computational Intelligence, volume 82, Springer, 2008, pp. 289–323.

[13] P. Angeline, D. Fogel, An evolutionary program for the identification of dynamical systems, in: Proceedings of SPIE Volume 3077: Application and Science of Artificial Neural Networks III., SPIE, 1997, pp. 409–417.

[14] D. Fogel, K. Chellapilla, Verifying anaconda's expert rating by competing against chinook: experiments in co-evolving a neural checkers player., Neurocomputing 42 (2002) 69–86.

[15] T. Froese, E. Spier, Convergence and crossover: the permutation problem revisited., Cognitive Science Research Papers CSRP 596 (2008).

[16] J. Schaffer, D. Whitley, L. Eshelman, Combinations of genetic algorithms and neural networks: A survey of the state of the art, in: Proceedings of the InternationalWorkshop on Combinations of Genetic Algorithms and Neural Networks, COGANN'92, IEEE Press, 1992, pp. 1–37.

[17] X. Yao, Evolving artificial neural networks, in: Proceedings of the IEEE, pp. 1423–1447.

[18] D. Thierens, J. Suykens, J. Vanderwalle, B. D. Moor, Genetic weight otpimization of a feedforward neural network controller., Artificial Neural Networks and Genetic Algorithms (1993) 658–663.

[19] F. Gomez, R. Miikkulainen, Active guidance for a fitness rocket through neuroevolution., in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'03, ACM, 2003, pp. 2084–2095.

[20] S. Haflidason, R. Neville, On the significance of the permutation problem in neuroevolution, in: Proc. of Genetic Evolutionary Computational Conference, GECCO'09, ACM, 2009, pp. 787–794.

[21] A. Mahmood, S. Sharmin, D. Barua, M. Islam, Graph matching recombination for evolving neural networks, in: Proc. of Int. Symposium on Neural Networks, ISNN'2007, Springer, 2007, pp. 562–568.

[22] M. Mandischer, Representation and evolution of neural networks, in: Artificial Neural Nets and Genetic Algorithms, Springer, 1993, pp. 643–649.

[23] N. Garcia-Pedrajas, D. Ortiz-Boyer, C. Hervas-Martinez, An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization, Neural Networks 19 (2006) 514–528.

[24] A. Azzini, A. Tettamanzi, Evolving neural networks for static single-position automated trading, Journal of Artificial Evolution and Applications 2008 (2008) 1–17.

[25] F. Camargo, Learning algorithms in neural networks, Technical Report, Computer Science Department, Columbia University, 1990.

[26] K. Gurney, An Introduction to Neural Networks, Taylor & Francis, Inc., Bristol, PA, USA, 1997.

[27] B. Kröse, P. Van der Smagt, An introduction to neural networks, URL ftp://ftp.informatik.uni-freiburg.de/papers/neuro/ann_intro_smag.ps.gz, The University of Amsterdam, 1996.

[28] G. Zhang, Neural networks for classification: A survey, IEEE Transaction on Systems, Man, and Cybernetics - Part C: Applications and Reviews 30 (2000) 451–462.

[29] A. Bahrammirzaee, A comparative survey of artificial intelligence applications in finance: artificial neural networks, expert system and hybrid intelligent systems, Neural Computing and Applications 19 (2010) 1165–1195.

[30] M. Castellani, H. Rowlands, Evolutionary artificial neural network design and training for wood veneer classification, Engineering Applications of Artificial Intelligence 22 (2009) 732–741.

[31] K. Ferentinos, Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms, Neural Networks 18 (2005) 934–950.

[32] J. Fernández, C. Hervás, F. Martínez-Estudillo, P. Gutiérrez, Memetic pareto evolutionary artificial neural networks to determine growth/no-growth in predictive microbiology, Applied Soft computing 11 (2011) 534–550.

[33] K. Tang, M. Lin, L. Minku, X. Yao, Selective negative correlation learning approach to incremental learning, Neurocomputing 72 (2009) 2796–2805.

[34] A. Azzini, A. Tettamanzi, A neural evolutionary approach to financial modeling, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'06, volume 2, Morgan Kaufmann, San Francisco, CA, pp. 1605–1612.

[35] A. Azzini, C. da Costa Pereira, A. Tettamanzi, Modeling turning points in financial markets with soft computing techniques, in: A. Brabazon, M. ONeill, D. Maringer (Eds.), Natural Computing in Computational Finance, volume 293 of *Studies in Computational Intelligence*, Springer Berlin / Heidelberg, 2010, pp. 147–167.

[36] A. Azzini, M. Lazzaroni, A. Tettamanzi, Incipient fault diagnosis in electrical drives by tuned neural networks, in: Instrumentation and Measurement Technology Conference, IMTC'06, pp. 1284–1289.

[37] A. Azzini, S. Marrara, Dermatology disease classification via novel evolutionary artificial neural network, in: Proceedings of the 18th International Conference on Database and Expert Systems Applications, DEXA'07, pp. 148–152.

[38] A. Azzini, A. Tettamanzi, A neural evolutionary classification method for brain-wave analysis, in: Proceedings of the European Workshop on Evolutionary Computation in Image Analysis and Signal Processing, EVOIASP'06, pp. 500–504.

[39] A. Azzini, A. Tettamanzi, Evolutionary ANNs: A state of the art survey, Intelligenza Artificiale 5 (2011) 19–35.

[40] T. Bäck, F. Hoffmeister, H. Schwefel, A survey of evolutionary strategies, in: R. Belew, L. Booker (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1991, pp. 2–9.

[41] H. Muhlenbein, D. Schlierkamp-Voosen, The science of breeding and its application to the breeder genetic algorithm (bga), Evolutionary Computation 1 (1993) 335–360.

[42] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Kluwer Academic Publishers, Boston, MA, 1989.

[43] H. Schwefel, Numerical Optimization for Computer Models, John Wiley, Chichester, UK, 1981.

[44] L. Prechelt, PROBEN1 — a set of neural network benchmark problems and benchmarking rules, Technical Report, Fakultät für Informatik, Universität Karlsruhe, 1994.

[45] M. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, Journal of Research of the National Bureau of Standards 49 (1952).

[46] A. Engelbrechts, Computational Intelligence: An Introduction, John Wiley & Sons, 2007.

[47] P. Gutiérrez, C. Hervás, M. Carbonero-Ruz, J. Fernández, Combined projection and kernel basis functions for classification in evolutionary neural networks, Neurocomputing 72 (2009) 2731–2742.

[48] F. Fernández-Navarro, C. Hervás-Martínez, P. Gutiérrez, M. Carbonero-Ruz, Evolutionary q-gaussian radial basis function neural networks for multiclassification, Neural Networks 24 (2011) 779–784.

[49] J.-Y. Wu, Mimo cmac neural network classifier for solving classification problems, Appl. Soft Comput. 11 (2011) 2326–2333.

[50] F. J. Martínez-Estudillo, C. Hervás-Martínez, P. Gutiérrez, A. C. Martínez-Estudillo, Evolutionary product-unit neural networks classifiers, Neurocomputing 72 (2008) 548–561.