Available online at www.sciencedirect.com
SCIENCE DIRECT
Neural Networks 19 (2006) 514–528
Neural Networks
www.elsevier.com/locate/neunet

# An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization

Nicolás García-Pedrajas *, Domingo Ortiz-Boyer, César Hervás-Martínez

*Department of Computing and Numerical Analysis, University of Córdoba, 14071 Córdoba, Spain*

Received 2 July 2004; accepted 11 August 2005

**Abstract**

In this work we present a new approach to crossover operator in the genetic evolution of neural networks. The most widely used evolutionary computation paradigm for neural network evolution is evolutionary programming. This paradigm is usually preferred due to the problems caused by the application of crossover to neural network evolution. However, crossover is the most innovative operator within the field of evolutionary computation.

One of the most notorious problems with the application of crossover to neural networks is known as the *permutation problem*. This problem occurs due to the fact that the same network can be represented in a genetic coding by many different codifications.

Our approach modifies the standard crossover operator taking into account the special features of the individuals to be mated. We present a new model for mating individuals that considers the structure of the hidden layer and redefines the crossover operator. As each hidden node represents a non-linear projection of the input variables, we approach the crossover as a problem on combinatorial optimization. We can formulate the problem as the extraction of a subset of near-optimal projections to create the hidden layer of the new network.

This new approach is compared to a classical crossover in 25 real-world problems with an excellent performance. Moreover, the networks obtained are much smaller than those obtained with classical crossover operator.


*Keywords:* Neural network evolution; Genetic algorithms; Crossover operator

## 1. Introduction

In the area of neural network design one of the main problems is finding suitable architectures for solving specific problems. The choice of such architecture is very important, as an inadequate network would be unable to learn or will end in over-fitting the training data. The problem of finding a suitable architecture and the corresponding weights of the network is a very complex task (Yao, 1999).

Evolutionary computation is a set of global optimization techniques that have been widely used in the last few years for training and/or automatically designing neural networks. There have been many applications for parametric learning (van Rooij, Jain, & Johnson, 1996) and for both parametric and structural learning (Yao, 1999; Yao & Liu, 1997; Angeline, Saunders, & Pollack, 1994; Odri, Petrovacki, & Krstonosic, 1993; Smalz & Conrad, 1994; Maniezzo, 1994; Moriarty & Miikkulainen, 1997), since Miller et al. (1991) proposed that evolutionary computation was a very good candidate to be used to search the space of topologies because the fitness function associated with that space is complex, noisy, non-differentiable, multi-modal and deceptive.

These works fall into two broad categories of evolutionary computation: genetic algorithms and evolutionary programming. The difference between these two categories is based upon two aspects: representation and variation operators. Representation, as well as variation operators, is standard in genetic algorithms and depends on the problem in evolutionary programming. An additional difference between genetic algorithms and evolutionary programming is the use of the crossover operator. This operator combines two or more individuals to produce a, possibly better, new individual. The benefits of crossover come from the ability to form connected substrings of the representation that correspond to above-average solutions (Goldberg, 1989a,b,c). These substrings are called *building blocks*. Crossover is not effective in environments where the fitness of an individual of the population is not correlated to the expected ability of its representational

* Corresponding author. Tel.: +34 957 211032; fax: +34 957 218630.
*E-mail addresses:* npedrajas@uco.es (N. García-Pedrajas), dortiz@uco.es (D. Ortiz-Boyer), chervas@uco.es (C. Hervás-Martínez).

0893-6080/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2005.08.014

components (Goldberg, 1989a,b,c). Such environments are called *deceptive* (Goldberg, 1989a,b,c).

Deception is a very important feature in most representations of neural networks, so crossover is usually avoided in evolutionary neural networks (Angeline et al., 1994). One of the most important forms of deception arises from the many-to-one mapping from genotypes in the representation space to phenotypes in the evaluation space. The existence of networks that are functionally equivalent and with different encodings makes evolution inefficient, and it is unclear whether crossover would produce more fitted individuals from two members of the population. This problem is usually termed as the *permutation problem* or the *competing conventions problem*. It is obvious that we can obtain a number of structurally different networks that implement the same input–output mapping but have different genetic representations.

For that reason, many evolutionary programming methods have been developed for evolving artificial neural networks, (Yao & Liu, 1997; Angeline et al., 1994; Moriarty & Miikkulainen, 1997; García-Pedrajas, Hervás-Martínez, & Muñoz-Pérez, 2002; García-Pedrajas, Hervás-Martínez, Muñoz-Pérez, 2003; Whitehead & Choate, 1996), but a comparatively small number of genetic algorithms have been used for neural network evolution. Nevertheless, removing the crossover operator has the disadvantage of removing the most innovative operator within the field of evolutionary computation.

In this paper, we propose a new approach for neural network crossover that is less sensitive to the permutation problem and improves the results of the evolved networks. Our aim is a redefinition of the crossover operator that avoids the deception problems stated above.

First of all, for this new approach to neural network crossover, we take into account the role of the hidden layer in a neural network. As it is stated in (Haykin, 1999):

> Hidden neurons play a critical role in the operation of a multilayer perceptron with back-propagation learning because they act as feature detectors. As the learning process progresses, the hidden neurons begin to gradually 'discover' the salient features that characterize the training data. They do so by performing a nonlinear transformation on the input data into a new space called the hidden space, or feature space. In this new space the classes of interest in a pattern-classification task, for example, may be more easily separated from each other than in the original input space.

Our approach to the genetic evolution neural networks takes into account this feature space. We center the evolution on the hidden node, as it is the most important feature of a neural network.

From this point of view, neural networks can be considered similar to *basis function* models (Denison, Holmes, Mallick, & Smith, 2002). These models assume that the function to be implemented, $g$, is made up of a linear combination of basis functions and corresponding coefficients. Hence $g$ can be written:

$$g(x) = \sum_{i=1}^{k} \beta_i B_i(x), \quad x \in \mathcal{X} \subset R^n, \tag{1}$$

where $\beta = (\beta_1,\ldots,\beta_k)'$ is the set of coefficients corresponding to basis functions $\boldsymbol{B} = (B_1,\ldots,B_k)$. Typically, the basis functions in (1) are non-linear transformations. Neural networks can be considered another example of basis function models. Methodologically, there is a major separation in neural network approaches, as the combination of the basis functions is not always linear, as in (1), and subsequent sets of basis functions, represented by hidden layers, can be constructed.

For neural networks, each hidden node can be considered as a basis function. Thus, a crossover operator will have the effect of exchanging a subset of basis functions between two networks or modifying some of the basis functions of the networks. Our redefinition of this operator tries to develop a process of exchanging hidden nodes meaningful from the point of view of the feature subspace.

Let us consider a feedforward neural network with $I$ inputs and a hidden layer with $H$ nodes. The hidden layer carries out a non-linear projection of input vector $\boldsymbol{x}$ to a vector $\boldsymbol{h}$ where:

$$h_i = f\left(\sum_{j=0}^{I} w_{ij} x_j\right). \tag{2}$$

As we have stated, each node performs a non-linear projection of the input vector. So, $\boldsymbol{h} = \boldsymbol{f}(\boldsymbol{x})$, and the output layer obtains its output from vector $\boldsymbol{h}$. In this context we can consider this projection as a basis function, so $B_i(x) = f\left(\sum_{j=0}^{I} w_{ij} x_j\right)$, and the output of the network is:

$$y(x) = F\left(\sum_{i=1}^{H} \beta_i B_i(x)\right) \tag{3}$$

where $F$ is the transfer function of the output layer, and the $\beta_i$ represents the weights of the connections from the hidden layer to the output layer. This projection performed by the hidden layer of a multi-layer perceptron distorts the data structure and inter-pattern distances (Lerner, Guterman, Aladjem, & Dinstein, 1999) in order to achieve a better classification. In the rest of the paper we will show how, from this point of view, we can redefine the crossover operator in a way that avoids several of its drawbacks.

This paper is organized as follows: Section 2 reviews the current approaches of genetic evolution of neural networks and their corresponding crossover operators; Section 3 explains the proposed new approach for neural network crossover; Section 4 describes the experimental setup; Section 5 shows the experimental results; and, finally, Section 6 states the conclusions of our work.

## 2. Crossover in genetic evolution of neural networks

Genetic algorithms are based on a representation that is independent of the problem. The representation is usually

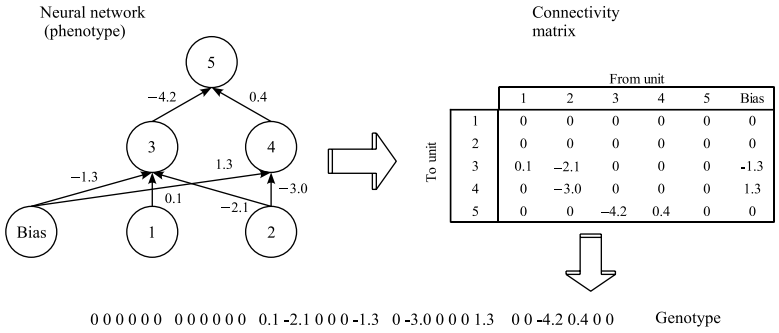


Fig. 1. Standard codification of a neural network by means of the linearization of the connectivity matrix for a real-coded genetic algorithm.

a string of binary, integer or real numbers. This representation (the genotype) codifies a network (the phenotype). This is a dual representation scheme. The interpretation function maps between the elements in the recombination space (on which the search is performed) and the subset of structures that can be evaluated as potential task solutions. The ability to create better solutions in a genetic algorithm relies mainly on the operation of *crossover*. This operator forms offspring by recombining representational components from two or more members of the population.

The most common approach for the genotype of the network consists of the linearization of the connectivity matrix (Balakrishnan & Honavar, 1995) (see Fig. 1).

Using this codification, standard crossover exchanges substrings between the two chromosomes. The result is two networks where the feature space has been randomly changed. In fact, this kind of crossover can be considered as a mutation rather than a crossover, as the offspring may have almost no behavioral connection with their parents.

Fig. 2 shows the effect on the phenotype of a node, the projection implemented by the node, after a uniform crossover operator in a two-dimensional space. We show the projection carried out by the two parents, and the corresponding projection carried out by the two offspring. We can see how the projections of the offspring have no connection with the projection of their parents.

Another alternative is placing the incoming and outgoing weights of a hidden node next to each other (Thierens, Suykens, Vandewalle, & Moor, 1991). The codification of the network of Fig. 1 using this model is shown in Fig. 3. Nevertheless, the experience of several authors suggests that it is important to treat connections from inputs to hidden nodes differently from connections from hidden nodes to outputs (Pujol & Poli, 1998).

Thierens (1996) presented an encoding that avoids the permutation problem, but even so, that encoding does not prevent the disruption of a favorable combination of weights. This coding avoids different representations for *exactly* the same network. The non-redundant codification transforms all the $2^H$ $H!$ equivalent networks, for a network of $H$ hidden nodes, to a *canonical* form.

However, the problem of finding building blocks in the distributed representation of a neural network is not alleviated by this codification. We still have the problem that very similar networks can have very different representations. A mutated network with just one different connection from its parent would have a completely different codification.

Moriarty and Miikkulainen (1997) developed a Marker-Based Encoding that resembles the DNA codification of

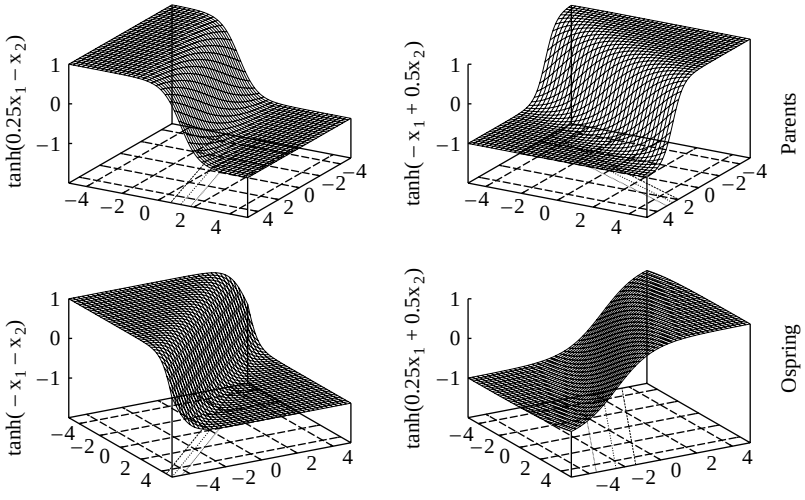


Fig. 2. Two nodes and their offspring using standard crossover. The figure shows how the projections on feature space of the offspring are very different from the projections of their parents.

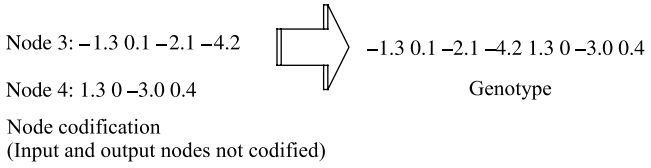


Fig. 3. Codification of a node placing incoming and outgoing weights of a node together.

chromosomes. Each network is defined by a string of integers ranging between −100 and 100. Some values of the range are defined as <start> symbols that mark the beginning of a node definition. Other values are defined as <end> symbols that mark the end of the node definition. The node is defined by the values between these two symbols. The connections of the node are codified as pairs <source/destination, weight>.

Using this codification, a standard two-point crossover is applied. Although the results are interesting in several applications (Moriarty & Miikkulainen, 1996, 1997), the effect of crossover on networks codified using the Marker-Based Encoding is usually very disruptive.

Although some successful models of genetically evolved neural networks can be found in the literature (Floreano & Urzelai, 2000; Cantú-Paz & Kamath, 2003), we think that these results could be improved if the specific functionality of the hidden layer is taken into account.

## 3. Combinatorial optimization approach to network crossover

As we have shown in the previous section, the drawbacks of the standard form of crossover for neural network evolution are numerous. So, we propose a new approach for network crossover based on the specific role of the hidden layer of a neural network in the computation carried out by the network.

In order to obtain an effective crossover operator, we must identify the *building blocks* of neural network evolution (Yao, 1999). Our basic assumption is that building blocks can be constituted by nodes or groups of nodes in the hidden layer. So, if we are able to effectively combine the hidden nodes of several networks we will obtain better offspring. In this way, we assume two working design restrictions as a starting point. The first restriction can be stated as:

**Restriction 1**. Each node is an indivisible unit of evolution. That is, no genetic operator can mate two nodes by mixing their internal structure.

This restriction is motivated by the fact that the mating of two nodes at this level produces something that is completely different from each one of its parents. Each node is a non-linear projection, the combination of the weights of two nodes will yield to a completely different projection. In this sense, such crossover operator is more a mutation operator. Following this restriction the $B_i(\boldsymbol{x})$ of (3) cannot be modified by a crossover operator.

The second design restriction of our work is:

**Restriction 2**. The combination of different projections constitute the 'building blocks' that must be preserved along the evolution.

The term 'projection' here is wider than the term 'node'. Different nodes in a network can represent the same projection, or very similar projections. In this way, the crossover operator must discard similar nodes.

The feature subspace that is obtained with the projection given by the hidden units facilitates the classification of the patterns. So, we consider that the combination of the projections of several networks can produce a better subspace for the classification of the given patterns.

More formally, we have $P$ parent networks, with $H_i$ hidden nodes, and with only one output without losing generality. Network $k$ is represented by:

$$y_j(x) = F\left(\sum_{i=1}^{H_1} \beta_{ji} B_{ji}(x)\right)$$

where $B_{ji} = f\left(\sum_{k=1}^{I} w_{ik}^{j} x_k\right)$ represents the $i$-th hidden node of network $j$, and $\beta_{ji}$ represents the connection from hidden node $i$ to the output in network $j$. As an offspring is a combination of some of the hidden nodes of these parents, we can define the output of the offspring, $y(\boldsymbol{x})$, as it was made up of all the hidden nodes of the parents, in the following way:

$$y(x) = F\left(\sum_{j=1}^{P} \sum_{i=1}^{H_j} \alpha_{ji} \beta_{ji} B_{ji}(x)\right), \tag{4}$$

where $\alpha_{ji} \in \{0, 1\}$ determines the presence/absence of each hidden node of the parent networks. As the basis functions are ordered, an offspring can be defined by the values of the $\alpha_{ji}$:

$$\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_P) = (0, 1, 0, 0, 0, 1, \ldots, 0). \tag{5}$$

In this way, the crossover operator can be defined as a combinatorial optimization problem. We have $P$ parents, each one being a network of the population. Every network $i$ has a hidden layer with $H_i$ hidden nodes. Each hidden node represents a non-linear projection of the input variables onto a new subspace. The crossover must obtain a subset of these $H = \sum_i H_i$ projections, of size $H_o$, $H_o \leq H$, that constitutes the optimal or near optimal combination of the $H$ projections. Defined in this way the problem is the optimal combination of $H$ elements. The whole process is depicted in Fig. 4.

Once we have established that the crossover operator is just a problem of combinatorial optimization, we can apply any algorithm that is efficient in solving this kind of problem. We have defined two types of crossover, using genetic algorithms and simulated annealing. These are two of the most successful algorithms for combinatorial optimization. Simulated annealing has been previously used for crossover operation in a problem of airspace traffic control (Delahaye & Odoni, 1997).

The proposed operator also has some reminiscences of the global discrete recombination in evolution strategies (Bäck,

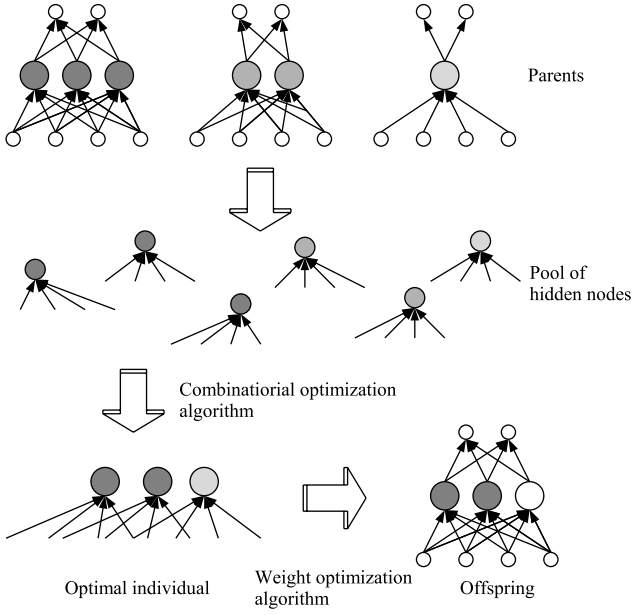


Fig. 4. Combinatorial optimization approach to crossover.

1996). This operator combines alleles of different parents to form an offspring.

First at all, we must state our problem formally. Let $\Omega = \{B_{11}(x), B_{12}(x), \ldots, B_{PH_P}(x)\}$, be the set of hidden nodes, $\Psi \subset \Omega$, and $\phi(\Psi) \geq 0$ a fitness function that assigns to any subset of $\Omega$ a value of its performance. Our problem can be stated as: *Optimize* $\phi(\Psi)$, where $\Psi \subset \Omega$. As $\Psi \subset \Omega$, our problem is to obtain the best subset of $\Omega$, using as a measure of fitness the function $\phi(\Psi)$. The actual form of this function must be defined in order to implement the optimization algorithm.

In order to define our combinatorial approach to network crossover, we must identify three aspects of this problem. These aspects are:

(1) The population of potential solutions that are subject to optimization, that is, the set $\Omega$.
(2) The algorithm for combinatorial optimization, that is, the method for obtaining the optimal subset of $\Omega$.
(3) The way of evaluating the individuals, that is, the actual form of $\phi(\cdot)$.

In the following three sections, we will explain in depth each one of these three elements of the crossover. We consider a crossover of $P$ parents, each one with $H_i$ hidden nodes, to obtain an offspring of $H_o$ number of nodes.

### 3.1. Pool of hidden nodes

In a first approach we can work with an $\Omega$ set formed by all the hidden nodes of all the parents, with a cardinality of $H_\Omega = \sum_{i=1}^{P} H_i$. However, this approach has two major drawbacks:

- The cardinality of $\Omega$ could be too high, making the combinatorial problem more difficult and the searching process slower. This is specially inappropriate, as the crossover must not overburden the evolutionary process.
- Some of the nodes of each network are not useful in the network. Its inclusion in $\Omega$ introduces more complexity without a compensation in terms of performance of the offspring.

In order to avoid these two problems we do not include all the nodes of all the parent networks in $\Omega$, but only the relevant nodes for each network are selected for the crossover. Nevertheless, measuring the relevance of a node within a network is not an easy task. Many methods have been developed for estimating the relevance of a hidden node within a network (Mozer & Smolensky, 1989; Le Cun, Denker, & Solla, 1990; Karnin, 1990; Hassibi & Stork, 1993), but most of them are too computationally expensive. The most straightforward method to estimate node relevance within a trained network consists of evaluating the effect that removing the node has over the error (Mozer & Smolensky, 1989). The network is evaluated with and without the node, and the following relevance measure for node $i$, $r_i$, is obtained:

$$r_i = \frac{\text{Perf}^k - \text{Perf}^k_{-i}}{\text{Perf}^k}, \tag{6}$$

where $\text{Perf}^k$ is the performance of the network $k$ and $\text{Perf}^k_{-i}$ is the performance of the network $k$ without the $i$-th node. All the nodes that fall below a certain threshold, $T_r$, of relevance are not included in the crossover operation. This threshold must be small enough to avoid removing nodes that are useful for the fine-tuning of the network. In our experiments we have set a threshold of $T_r = 0.01$, that is, only the nodes that account for less than the 1% of the performance of the network are removed.

### 3.2. Node combination evaluation

The evaluation of the subset of hidden nodes is not as straightforward as the evaluation of a whole network. In neural network evolution, Miller et al. (1991) identified two approaches: the *strong specification scheme*, where each connection of the network is specified by its numerical representation, and the *weak specification scheme*, where the exact structure of the network is not explicitly codified but is computed based on the information contained in the chromosome. The former has been more widely used, as the latter introduces more noise in the fitness function and is more deceptive, due to the fact that there is a one-to-many mapping between the space of representation and the solution space.

The evaluation of a subset of nodes can be considered similar to the *weak specification scheme*, as we need to train an output layer before being able to evaluate the performance of the network. In order to get an accurate estimation of the performance of the network, the procedure would need to evaluate the network with several initializations of the weights of the output node. However, this procedure would be too computationally expensive. So, we train the output layer once, and, in order to avoid the noise of a random initialization of the weights of the output layer, we use the outputs weights of each

node within its original network as initial weights before applying the backpropagation algorithm.

Once the final individual is obtained, we apply an additional backpropagation algorithm to set the output weights of the offspring network. In order to make a fair comparison, this step is also performed for the standard crossover.

### 3.3. Combinatorial optimization algorithm

Once we have stated our problem, there are many algorithms that could be applied to its solution. Among the most widely used algorithms for combinatorial optimization are simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), genetic algorithms (Michalewicz, 1994), particle swarm optimization (Kennedy & Eberhart, 1995) and ant colonies (Dorigo, Maniezzo, & Colorni, 1996). However, we have a prerequisite to be met by any algorithm to be useful in our crossover: it must not be computationally expensive. On the other hand, it must also be effective in finding good optima. Mixing these two conditions, the most suitable approaches are simulated annealing and genetic algorithms, the latter with a very small population.

#### 3.3.1. Simulated annealing crossover

Simulated annealing is one of the most common methods for solving combinatorial problems. We have chosen this algorithm because it is easy to implement and its computational cost is moderated. For stating our simulated annealing algorithm, we must define the value of the temperature, the initial solution, the way of obtaining the next solution, and the fitness function. Although it has been suggested that the final result is greatly affected by these decisions (Hajek, 1988), we have always preferred the fastest option, in order not to overburden our evolutionary process.

The temperature is obtained using $T(t+1)=\delta T(t)$, $T(0)=T_o$. There are many other temperature schedules, but there is no significant difference in their performance, provided they generate temperatures with comparable speed.

The initial solution is chosen selecting $n$ hidden nodes randomly, where $n$ is the average number of hidden nodes of the parents. The current solution is codified as a binary vector, and each element of the vector codifies whether the corresponding node of the pool is present/absent in the current solution. So, we assign an index to every hidden node in $\Omega$, and each individual is a binary vector of length $H_\Omega = card(\Omega)$.

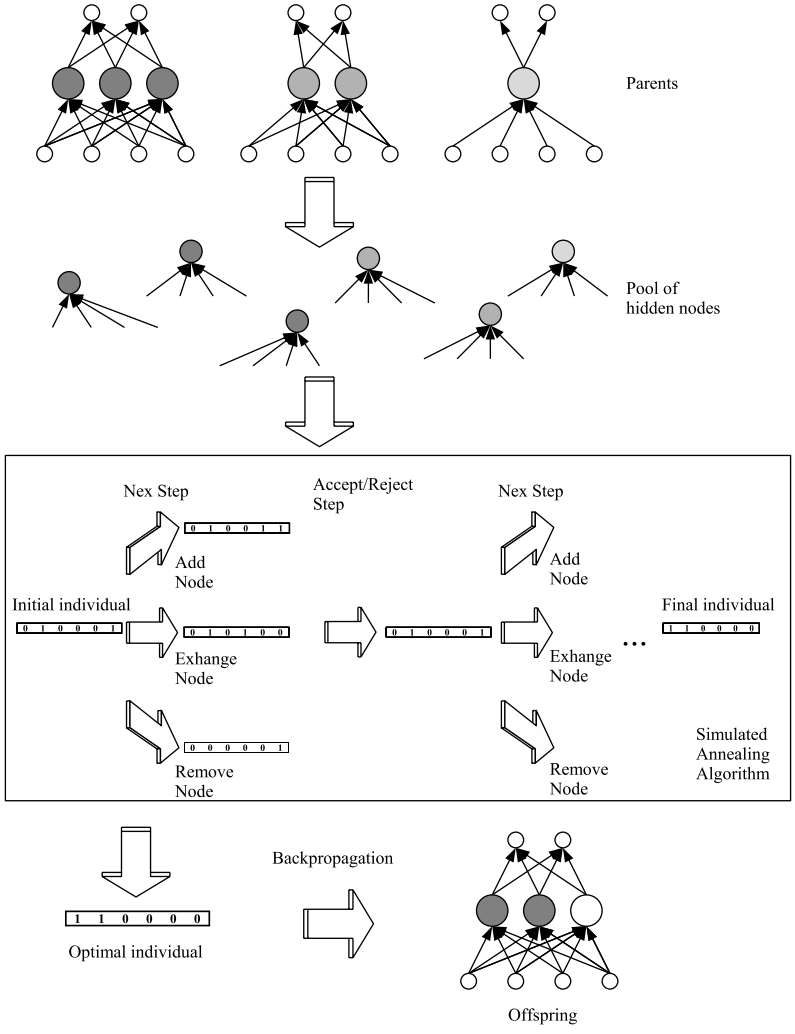


Fig. 5. Simulated annealing based crossover.

The value of a position of the vector is 1 if the corresponding hidden node belongs to the individual, and 0 otherwise. This representation, where each individual is a string of bits that correspond to the vector $\boldsymbol{\alpha}$, follows directly from (5).

Theoretical results show that, in order to guarantee the success of the search process, it is necessary for any solution to be reachable from any other solution by means of a valid sequence of steps. In order to achieve this condition, the random step in every iteration of the algorithm consists of a random selection among three different moves: addition of a node from the pool of nodes, deletion of a node, and exchange of a node with the pool of nodes. The three possible moves are applied with equal probability. The steps of the simulated annealing crossover are summarized in Algorithm 1.

The evaluation of each solution is carried out following the method explained in Section 3.2. This process is shown in Fig. 5.

**Algorithm 1: Simulated annealing based crossover.**

**Data**: $P$ parent networks
**Result**: An offspring network
Initialize pool of hidden nodes;
Initialize and evaluate initial solution;
old fitness = 0;
$T = T_o$;
**for** i = 1 to iterations **do**
"step" = random {Add node, remove node, exchange node };
Perform "step";
new_fitness = Evaluation of new solution;
**if** new_fitness > old_fitness or random(0, 1) < $e^{-(\text{old\_fitness} - \text{new\_fitness})/T}$ then
Accept new solution;
else
Reject solution;
end
$T = \delta T$;
end
Train output layer of final solution;

*3.3.2. Genetic algorithm crossover*

The implemented genetic algorithm for the crossover operator is as simple as possible. Our objective is obtaining the best subset of $\Omega$. The population is small and it is evolved for a small number of generations. The representation of the individuals is the same used in the simulated annealing, each individual is a string of bits of length $H_\Omega$ representing the presence/absence of each hidden node.

For the evolution of the population, we have used the CHC approach, because it has obtained very good results with small populations (Eshelman, 1990). CHC stands for *Cross generational elitist selection, Heterogeneous recombination and Cataclysmic mutation*. The non-traditional CHC genetic algorithm differs from traditional GAs in a number of ways (Louis & Li, 1997):

(1) To obtain the next generation for a population of size $N$, the parents and the offspring is put together and the $N$ best individual are selected.
(2) To avoid premature convergence, only different individuals, separated by a threshold Hamming distance— in our implementation 4 bits— are allowed to mate.
(3) During crossover, two parents exchange exactly half of their non-matching bits. This operator is called *Half Uniform Crossover* (HUX) crossover (Eshelman, 1990).
(4) Mutation is not used during the regular evolution. In order to avoid premature convergence or stagnation of the search, the population is reinitialized when the individuals are not diverse. In such a case only the best individual is kept in the new population.

The procedure for carrying out a crossover using a genetic algorithm is outlined in Fig. 6. First of all, we obtain the members of $\Omega$, which are all the hidden nodes of the parent networks whose relevance is above the given threshold. With this pool of hidden nodes, we perform the genetic algorithm. The best individual is selected and a final backpropagation training is carried out in order to allow all the output connections of the hidden nodes to adapt to its new companions. The process is shown in Algorithm 2.

**Algorithm 2: Genetic algorithm based crossover**

**Data**: $P$ parent networks
**Result**: An offspring network
Initialize pool of hidden nodes;
Initialize and evaluate population;
**for** i = 1 to generations **do**
**for** j = 1 to population_size/2 **do**
Select two random individuals, $i_1$ and $i_2$, without replacement;
**if** Hamming distance ($i_1$, $i_2$) > 4 **then**
Perform HUX crossover;
Add offspring to new population;
end
end
Evaluate new individuals;
Select best population_size individuals as new population;
end
Train output layer of final solution;

*3.4. Permutation problem*

Permutation problem can harm the evolutionary process making crossover operator inefficient. Networks that have several similar hidden nodes will likely produce poor offspring when they mate. The proposed crossover does not remove the permutation problem, as this problem is associated with the

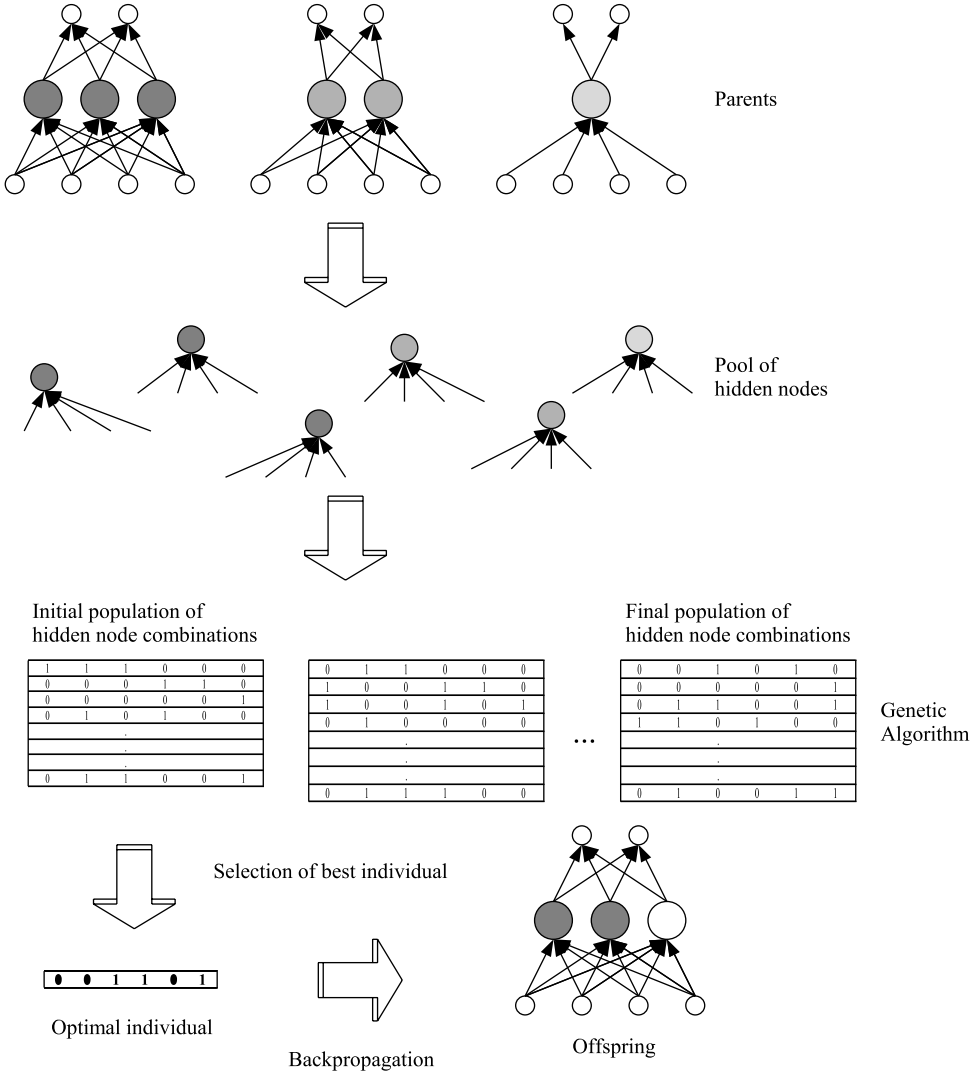


Fig. 6. Genetic algorithm based crossover.

representation of the networks, but can greatly alleviate most of its harmful effects.

We can identify three fundamental effects of the permutation problem that can harm the efficiency of the crossover. These scenarios are the result of having different networks with the same, or almost the same, function and different codification. In the extreme case let us consider two networks, $n_1$ and $n_2$, each one having the same four nodes, $a$, $b$, $c$, $d$, but arranged in different order, $n_1 = \{a, b, c, d\}$ and $n_2 = \{c, d, a, b\}$. Each letter represents a node with a different functionality. If these two networks mate using a standard one-point crossover, they may produce two offspring:

$$n_{12} = \{a, b, a, b\}$$

$$n_{21} = \{c, d, c, d\}$$

The first negative effect is that the offspring will perform worse than the parents (Angeline et al., 1994), as it will lack key computational components of its parents. In our method, all the nodes of the two parents can be inherited by the offspring, so the potential ability of the offspring is, at least, the same as that of its parents.

The second negative effect is that each representation is another convention of the same solution (Schaffer, Whitley, & Eshelman, 1992), so it is an extra region in the search space. This effect is greatly attenuated by our model. When two networks mate, the order of their nodes does not matter, so if they have nodes with functionalities that are useful together, there is a high probability that the combinatorial optimization algorithm will find such useful combination.

If in an iteration of the optimization algorithm an individual is formed by $\{a, b, a, b\}$, the combinatorial optimization algorithm and the regularization term will remove the redundant node. After all the iterations of any of the two combinatorial optimization algorithms proposed, we think that the probability of obtaining a final individual with two redundant nodes is almost negligible. Moreover, using more aggressive selection reduces the risk of exploring several competing conventions (Branke, 1995), and in a sense, the pressure of the combinatorial optimization algorithm fitness

and regularization terms is very similar to a high selective selection in the genetic algorithm.

Even if similar nodes are inherited by a descendant, the relevance threshold imposed on each node to be part of the crossover operation will prevent the subsequent inheritance of these two similar nodes.

The third negative effect is the breaking of the *behavioral link* between the parents and their offspring. Two networks with nodes $\{a, b, a, b\}$ and $\{c, d, c, d\}$ will be likely to perform very differently from their parents. In our crossover, the behavioral link is enforced. The offspring can inherit all the useful nodes of its parents, so the useful functionality of the parents is also likely to be inherited.

### 3.5. Evolutionary process

The evolution of the population of networks is as standard as possible. The best $P_e$% individuals of the population are replicated, the rest of the new population is obtained by crossover. The new population is subject to random mutation, with $P_r$ probability, and backpropagation mutation, with $P_{bp}$ probability, in this order.

The fitness of each individual is composed of two terms, a performance term and a regularization term. The performance term, $p$, is the number of training patterns correctly classified by the individual. The regularization term, $r$, is just the number of nodes. Other more sophisticated terms can be taken from the literature (Setiono, 1997) (Hinton, 1993), but they are computationally expensive and their effect over the performance of the model is still under discussion.

On the other hand, several studies (Lawrence, Giles, & Tsoi, 1996; Weigend, 1993; Caruana, Lawrence, & Giles, 2001) have concluded that overtraining is a problem of both small and large networks. So, our regularization term is intended to penalize very large networks and hardly affects moderately large and small networks. As the fitness value is within the interval [0, 1], the regularization term must has a moderate effect over small networks, and its relevance must be increased as the size of the networks grows above small values. In this way, the fitness of individual $i$, $f_i$, is given by:

$$f_i = p_i - \frac{1}{2} r_i, \tag{7}$$

where $r_i = e^{0.0069 H_i} - 1$, being $H_i$ the number of nodes of network $i$, and $p_i$ is just the number of patterns of the training set correctly classified divided by the total number of training patterns.

The stop criterion, for all the evolutionary algorithms used in this paper, is the stagnation of the fitness of the population. The evolution ends if, during the last 10 generations, the average fitness of the population does not improve above a certain threshold. In all the experiments this threshold is set to 10%.

### 3.6. Node mutation

As in other models of evolutive networks, we have two types of mutation: parametric and structural mutation. Parametric mutation is carried out in order to adjust the weights of the evolved networks. Without parametric mutation it is very difficult for the evolutive process to fine-tune its weights. In our model we considered two standard parametric mutation operations:

- *Backpropagation mutation.* The network is trained using a simple back-propagation algorithm for a fixed number of cycles. We use cross-validation and early-stopping. The validation set is taken randomly from the training set every time the back-propagation algorithm is run.
- *Random mutation.* A small quantity, normally distributed with zero mean and a small standard deviation, is added to all the weights of the network.

These two simple mutations are chosen in an attempt to minimize the effect of mutation on the model, for a fair comparison of the proposed crossover methodology and the standard methodology of crossover. In the first set of experiments, we did not use structural mutation, as we wanted to compare the performance of the different crossover operators with the least possible influence of other aspects of the evolution.

In summary, crossover based on combinatorial optimization has the following advantages over standard crossover for neural networks:

- The order of a node within a network is not relevant. This fact avoids the consequences of the permutation problem.
- Two non-contiguous nodes in a network can be inherited together. In this way, the potential number of networks that can be the offspring of a crossover is dramatically increased. In our approach, given that the cardinality of $\Omega$ is $H_\Omega$, the number of potential offspring is $\sum_{n=1}^{H_\Omega} (H_\Omega / n)$. In a standard two-point crossover, with two parents of length $n$ and $m$, $n < m$, the upper bound for the number of possible offspring is $n^2$, a value clearly below the former.
- The evaluation of the individuals during the crossover takes into account the regularization term, so, redundant nodes are more likely to be removed during this process.

## 4. Experimental setup

In order to make a fair comparison among the different methods for the crossover operator, all the experiments were carried out using the same parameter set. The population has 100 networks with a maximum of hidden nodes of $H_{max} = 100$. Elitism is $P_e = 0.1$, the probability of random and backprop mutation is $P_{bp} = 0.25$ and $P_r = 0.25$, respectively. For backprop algorithm $\eta = 0.15$ and $\alpha = 0.15$. For combinatorial crossover we used five parents, and a relevance threshold of 1%. The simulated annealing algorithm was run 1000 steps. The population of the genetic algorithm crossover was of 25 individuals and was evolved for 20 generations. All the algorithms are programmed in C and the code is available upon request to the authors.

The initialization of the networks was made using the method proposed in (Le Cun, 1993). This method was applied to the initialization of the networks in all the experiments. The number of hidden nodes, $H$, is randomly selected in the interval $H \in [1, H_{\max}]$. The weight of each connection is randomly selected from $\left[-\sqrt{3}/n_i, \sqrt{3}/n_i\right]$, $n_i$ being the number of inputs to the node.

Each set of available data was divided into two subsets: 75% of the patterns were used for learning, and the remaining 25% for testing the generalization of the networks. We have used 25 datasets from the UCI Machine Learning Repository that cover a wide variety of problems. Testing our model on this problems can give us a clear idea of its performance.

The comparisons among the performance of the different methods were made by means of $t$-tests. In all the tables the $p$-values of the corresponding tests are shown. For each experiment, the execution was repeated 30 times with different random seeds. From each experiment the result of the evolution was the smallest network from the 10 best of the population. In this way, we try to avoid the possibly overtrained best network. This method of selecting the best network was applied to all the experiments, regardless of other parameters.

## 5. Experimental results

In the first series of experiments, we wanted to test the performance of the model in solving the given classification problems. This performance is compared with the performance obtained by the evolutive process using the standard crossover. The results obtained with these models are shown in Table 1. In all the tables, and in the following discussion, we will use StdX for standard crossover, GaX for the crossover based on the genetic algorithm, and SaX for the crossover based on simulated annealing. The error measure is $E = \frac{1}{N}\sum_{i=1}^{N} e_i$, where $e_i$ is 1 if pattern $i$ is misclassified and 0 otherwise, and $N$ is the number of patterns of the corresponding set. In all the tables we show the average error and the standard deviation. We also show the $p$-values of the $t$-test.

From Table 1 we can see how the proposed crossover obtains better results than the standard crossover. For 20 of the 25 problems the performance of the crossover based on a genetic algorithm is better than the standard crossover, the difference being statistically significant at a confidence level of 0.05. Of the other five, in two of them, labor and zoo, the performance of all the models is almost of 100%, so it can hardly be improved. For iris and sonar the performance of GaX is better but the difference is not statistically significant. Finally, for glass the performance of GaX and StdX is the same. The performance of SaX is slightly worse, SaX performs better than StdX in 18 of the 25 problems.

If we compare the two crossovers based on combinatorial optimization, the performance of GaX is, in general, above the performance of SaX. This result enforces the idea underlying our approach. If the combinatorial optimization is a good idea, a crossover based on a genetic algorithm should obtain better results than a crossover based on simulated annealing, as the former is a better combinatorial optimization algorithm than the latter. We must note that the parameters of the two

Table 1
Results for the classification of the described problems

| Problem | StdX | | GaX | | SaX | | $t$-test ($p$-values) | | |
|---|---|---|---|---|---|---|---|---|---|
| | Mean | SD | Mean | SD | Mean | SD | StdX-GaX | StdX-SaX | GaX-SaX |
| Anneal | 0.1019 | 0.0332 | *0.0665* | 0.0295 | *0.0779* | 0.0256 | 0.0000 | 0.0014 | 0.0575 |
| Autos | 0.3810 | 0.0638 | *0.3373* | 0.0693 | *0.3240* | 0.0769 | 0.0065 | 0.0010 | 0.3962 |
| Balance | 0.0923 | 0.0122 | *0.0821* | 0.0099 | *0.0829* | 0.0135 | 0.0007 | 0.0063 | 0.7801 |
| Cancer | 0.0619 | 0.0347 | *0.0046* | 0.0038 | 0.0248 | 0.0217 | 0.0000 | 0.0000 | 0.0000 |
| Card | 0.1390 | 0.0105 | *0.1302* | 0.0084 | *0.1318* | 0.0078 | 0.0008 | 0.0040 | 0.4640 |
| Gene | 0.1704 | 0.0132 | *0.1393* | 0.0104 | *0.1388* | 0.0096 | 0.0000 | 0.0000 | 0.8279 |
| German | 0.2645 | 0.0225 | *0.2507* | 0.0120 | *0.2537* | 0.0147 | 0.0035 | 0.0318 | 0.3592 |
| Glass | 0.3465 | 0.0542 | 0.3466 | 0.0432 | 0.3556 | 0.0911 | 0.9955 | 0.6237 | 0.6029 |
| Glass-g2 | 0.0717 | 0.0183 | *0.0490* | 0.0111 | *0.0533* | 0.0243 | 0.0000 | 0.0017 | 0.3772 |
| Heart-c | 0.1333 | 0.0322 | *0.1154* | 0.0316 | *0.1092* | 0.0235 | 0.0331 | 0.0016 | 0.3965 |
| Heart-s | 0.1436 | 0.0246 | *0.1230* | 0.0170 | *0.1275* | 0.0209 | 0.0004 | 0.0081 | 0.3742 |
| Hepatitis | 0.1105 | 0.0341 | *0.0930* | 0.0179 | *0.0939* | 0.0246 | 0.0156 | 0.0342 | 0.8752 |
| Horse | 0.2846 | 0.0385 | *0.2656* | 0.0265 | 0.2780 | 0.0216 | 0.0295 | 0.4172 | 0.0506 |
| Hypothyroid | 0.0559 | 0.0029 | *0.0509* | 0.0013 | 0.0522 | 0.0017 | 0.0000 | 0.0000 | 0.0016 |
| Ionosphere | 0.1134 | 0.0229 | *0.1011* | 0.0174 | *0.1034* | 0.0132 | 0.0233 | 0.0435 | 0.5667 |
| Iris | 0.0324 | 0.0165 | 0.0284 | 0.0060 | 0.0295 | 0.0080 | 0.2019 | 0.3688 | 0.4712 |
| Krvs.kp | 0.0496 | 0.0057 | *0.0383* | 0.0035 | 0.0442 | 0.0049 | 0.0000 | 0.0002 | 0.0000 |
| Labor | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.0000 | 1.0000 | 1.0000 |
| Lymphography | 0.2297 | 0.0610 | *0.1586* | 0.0272 | *0.1568* | 0.0385 | 0.0000 | 0.0000 | 0.8349 |
| Pima | 0.2082 | 0.0221 | *0.1840* | 0.0105 | 0.1939 | 0.0196 | 0.0000 | 0.0106 | 0.0178 |
| Sick | 0.0515 | 0.0048 | *0.0486* | 0.0019 | 0.0508 | 0.0036 | 0.0038 | 0.5218 | 0.0051 |
| Sonar | 0.1946 | 0.0171 | 0.1888 | 0.0221 | 0.1885 | 0.0256 | 0.2622 | 0.3105 | 0.9713 |
| Vote | 0.0599 | 0.0139 | *0.0522* | 0.0107 | 0.0546 | 0.0140 | 0.0194 | 0.1514 | 0.4473 |
| Vowel | 0.6188 | 0.0452 | *0.5901* | 0.0434 | *0.5938* | 0.0404 | 0.0150 | 0.0419 | 0.7521 |
| Zoo | 0.0280 | 0.0186 | 0.0291 | 0.0182 | 0.0282 | 0.0188 | 0.8063 | 0.9561 | 0.8231 |

For each test we show the average test error and the standard deviation of the test error. The best result for each problem is shown in italics

algorithms are chosen with the idea that the number of evaluations of the fitness function on both methods would be the same.

The networks obtained with the proposed model are smaller than those obtained with the standard crossover (for detailed results see Table 2). In fact, they are among the smallest obtained in the literature (Yao & Liu, 1997; Setiono, 2001). The size of the networks is an interesting consequence of the combinatorial optimization approach. As the size of a network is part of its fitness, the combinatorial approach will always prefer smaller networks, avoiding the addition of superfluous hidden nodes.

In order to get a clear idea of the comparative size of the obtained networks, Table 2 also shows the average size of the networks obtained by previous works for the same problems. We can see that the networks obtained by combinatorial optimization are among the smallest ones.

### 5.1. Comparison with the evolution without crossover

Many papers rely on mutation as the only operator for evolving neural networks. In order to test whether the inclusion of crossover, in any of its forms, has a beneficial effect over the performance of the evolved networks, we have carried out all the evolutions without using crossover. The best $P_e$% is replicated, the rest of the new population is obtained by mutation. The new population is subject to structural mutation, with $P_s$ probability, random mutation, with $P_r$ probability, and backpropagation mutation, with $P_{bp}$ probability, in that order.

We have introduced a structural mutation for compensating the lack of crossover operation. Otherwise, we thought that the comparison would not be fair. The probability of this structural mutation in the experiments, $P_s$, was of $P_s = 0.25$. The structural mutation consists of randomly selecting one of the four following operations: deletion/addition of a node, and deletion/addition of a connection.

The study is made for all the data sets. We wanted to test the efficiency of each crossover operator, studying the effect over the network when it is removed. The detailed results are shown in Table 3. For each problem we show the averaged generalisation error obtained, when no crossover is used. The *t*-test compares the error means of each crossover operator and the experiment without crossover.

These results suggest several interesting conclusions about the role of the crossover in the evolution of neural networks. On the one hand, we can see that the standard crossover is useful in some problems, but it has no significant effect on 10 of the 25 problems. On the other hand, combinatorial based crossovers are useful in 24 of the 25 problems. We can conclude that the use of the standard crossover is useful in many classification problems, and that our approach is more efficient in improving the performance of the obtained networks.

### 5.2. Effect of the regularization term

Some studies (Lawrence et al., 1996; Weigend, 1993; Caruana et al., 2001; Weigend, 1994) have shown that overtraining is an effect that appears not only in large networks but

Table 2
Average size of the networks obtained with the three methods and of networks obtained in other papers

| Problem | Size (nodes) | | | *t*-test (*p*-values) | | | Size of other methods | | |
|---|---|---|---|---|---|---|---|---|---|
| | StdX | GaX | SaX | StdX-GaX | StdX-SaX | GaX-SaX | (Setiono, 2001) | (Treadgold & Gedeon, 1999) | (Yang, Parekh, & Honavar, 1997) |
| Anneal | 8.40 | 2.52 | 3.00 | 0.0000 | 0.0000 | 0.0000 | 6.58 | – | – |
| Autos | 7.80 | 7.55 | 4.43 | 0.7209 | 0.0000 | 0.0000 | 8.38 | – | – |
| Balance | 3.07 | 2.00 | 3.40 | 0.0000 | 0.2794 | 0.0000 | 10.12 | – | 12 |
| Cancer | 1.13 | 1.00 | 1.00 | 0.0390 | 0.0390 | 1.0000 | 3.40 | 4.86 | 14 |
| Card | 3.33 | 1.00 | 1.00 | 0.0000 | 0.0000 | 1.0000 | 7.22 | 0.12 | – |
| Gene | 5.50 | 2.30 | 2.14 | 0.0000 | 0.0000 | 0.0995 | – | 0.00 | – |
| German | 3.33 | 1.00 | 1.00 | 0.0001 | 0.0001 | 1.0000 | 9.54 | – | – |
| Glass | 8.87 | 7.85 | 7.54 | 0.0000 | 0.0015 | 0.0528 | 9.58 | 4.18 | 27 |
| Glass-g2 | 3.73 | 3.62 | 2.80 | 0.6100 | 0.0001 | 0.0001 | 7.06 | – | – |
| Heart-c | 3.37 | 1.53 | 2.20 | 0.0000 | 0.0064 | 0.0000 | 5.60 | 0.20 | – |
| Heart-s | 4.23 | 1.50 | 1.63 | 0.0000 | 0.0000 | 0.5175 | 5.94 | 0.40 | – |
| Hepatitis | 3.53 | 1.00 | 1.57 | 0.0000 | 0.0000 | 0.0001 | 5.66 | – | – |
| Horse | 8.00 | 2.17 | 2.17 | 0.0000 | 0.0000 | 1.0000 | 4.88 | 0.12 | – |
| Hypothyroid | 1.60 | 1.07 | 1.00 | 0.0018 | 0.0003 | 0.1555 | 10.02 | 4.64 | – |
| Ionosphere | 3.53 | 1.00 | 1.20 | 0.0000 | 0.0000 | 0.0779 | 4.20 | – | 10 |
| Iris | 3.13 | 2.00 | 2.05 | 0.0000 | 0.0000 | 0.1753 | 3.34 | – | – |
| Krvs.kp | 2.87 | 1.00 | 1.00 | 0.0000 | 0.0000 | 1.0000 | 7.20 | – | – |
| Labor | 1.23 | 1.00 | 1.00 | 0.0139 | 0.0139 | 1.0000 | 2.46 | – | – |
| Lymphography | 5.60 | 1.57 | 1.90 | 0.0000 | 0.0000 | 0.0030 | 6.34 | – | 72 |
| Pima | 4.63 | 1.00 | 1.00 | 0.0000 | 0.0000 | 1.0000 | 5.88 | 3.02 | – |
| Sick | 1.90 | 1.00 | 1.00 | 0.0000 | 0.0000 | 1.0000 | 4.80 | – | – |
| Sonar | 7.43 | 1.00 | 3.13 | 0.0000 | 0.0000 | 0.0000 | 5.18 | – | – |
| Vote | 1.10 | 1.00 | 1.07 | 0.0779 | 0.7026 | 0.3215 | 4.42 | – | – |
| Vowel | 8.57 | 7.27 | 7.26 | 0.0000 | 0.0000 | 0.9662 | 17.44 | – | 55 |
| Zoo | 8.27 | 5.27 | 5.18 | 0.0000 | 0.0000 | 0.3919 | 12.12 | – | – |

Table 3
Results for the classification without using crossover

| Problem | Generalization | | | | *t*-test (*p*-values) | | |
|---|---|---|---|---|---|---|---|
| | Mean | SD | Best | Worst | StdX | GaX | SaX |
| Anneal | 0.1792 | 0.0571 | 0.0893 | 0.3795 | 0.0000 | 0.0000 | 0.0000 |
| Autos | 0.4523 | 0.0663 | 0.3333 | 0.6078 | 0.0001 | 0.0000 | 0.0000 |
| Balance | 0.1045 | 0.0131 | 0.0641 | 0.1282 | 0.0004 | 0.0000 | 0.0000 |
| Cancer | 0.0931 | 0.0440 | 0.0230 | 0.2069 | 0.0034 | 0.0000 | 0.0000 |
| Card | 0.1351 | 0.0103 | 0.1163 | 0.1628 | *0.1548* | *0.0513* | *0.1690* |
| Gene | 0.2410 | 0.0216 | 0.1917 | 0.2837 | 0.0000 | 0.0000 | 0.0000 |
| German | 0.2759 | 0.0204 | 0.2320 | 0.3200 | 0.0458 | 0.0000 | 0.0000 |
| Glass | 0.4113 | 0.0765 | 0.3019 | 0.6226 | 0.0004 | 0.0001 | 0.0097 |
| Glass-g2 | 0.0917 | 0.0396 | 0.0250 | 0.1750 | 0.0148 | 0.0000 | 0.0000 |
| Heart-c | 0.1355 | 0.0358 | 0.0921 | 0.2237 | *0.8037* | 0.0242 | 0.0013 |
| Heart-s | 0.1490 | 0.0278 | 0.0882 | 0.2059 | *0.4296* | 0.0001 | 0.0012 |
| Hepatitis | 0.1202 | 0.0350 | 0.0526 | 0.1842 | *0.2843* | 0.0004 | 0.0014 |
| Horse | 0.3011 | 0.0382 | 0.2308 | 0.3846 | *0.1027* | 0.0001 | 0.0056 |
| Hypothyroid | 0.0622 | 0.0045 | 0.0551 | 0.0753 | 0.0000 | 0.0000 | 0.0000 |
| Ionosphere | 0.1203 | 0.0183 | 0.0920 | 0.1609 | *0.2030* | 0.0001 | 0.0001 |
| Iris | 0.0360 | 0.0148 | 0.0270 | 0.0811 | *0.3764* | 0.0312 | 0.0092 |
| Krvs.kp | 0.0530 | 0.0051 | 0.0438 | 0.0638 | 0.0172 | 0.0000 | 0.0000 |
| Labor | 0.0357 | 0.0363 | 0.0000 | 0.0714 | 0.0000 | 0.0000 | 0.0000 |
| Lymphography | 0.2532 | 0.0686 | 0.1351 | 0.3784 | *0.1675* | 0.0000 | 0.0000 |
| Pima | 0.2142 | 0.0219 | 0.1823 | 0.2552 | *0.2891* | 0.0000 | 0.0004 |
| Sick | 0.0550 | 0.0032 | 0.0488 | 0.0615 | 0.0013 | 0.0000 | 0.0000 |
| Sonar | 0.2388 | 0.0317 | 0.1827 | 0.3077 | 0.0000 | 0.0000 | 0.0000 |
| Vote | 0.0623 | 0.0106 | 0.0370 | 0.0741 | *0.4422* | 0.0005 | 0.0194 |
| Vowel | 0.6725 | 0.0473 | 0.5801 | 0.7814 | 0.0000 | 0.0000 | 0.0000 |
| Zoo | 0.0680 | 0.0422 | 0.0000 | 0.2000 | 0.0000 | 0.0000 | 0.0000 |

For each test we show the average test error, the standard deviation, and the best and worst results. The experiments where the crossover operator does not significantly improve the results, for a significant level of 0.05, are shown in italics.

also in small networks. So, the use of a regularization term is under study, as small networks can also suffer from over-training and it is more likely for a small network to be unable to learn. In order to test the effect of the regularization term, we considered two regularization coefficients: $X$ and $All$. $X$ is the regularization coefficient applied when the combinatorial optimization is being carried out. $All$ is the regularization coefficient applied when the individuals of the population are being evaluated.

In order to test the effect of regularization we evolved the population using the four possible combinations of the two previous coefficients. We carried out 10 runs for each number of parents. As the experiment with $X=All=1$ coincides with the initial experiment, we took the first 10 runs of this experiment.

The study is carried out for six of the above studied problems: glass-g2, heart-c, heart-s, hepatitis, lymphography, and pima. As the behavior of the model is similar using both simulated annealing and genetic algorithms for the crossover, in this study we will only use the later in order to reduce the number of experiments. The results are shown in Table 4.

From the values shown in the table, we can extract some conclusions about the effect of the regularization term on the evolution of the networks:

- The effect of the regularization is positive. When we remove this term, the networks grow bigger and their generalization error does not significantly decrease, and even, for some problems, increases.
- Using the regularization term during the combinatorial optimization is efficient for keeping the networks small, even if the regularization is not used during the evaluation of the individuals.

Table 4
Results for the classification of glass-g2, heart-c, heart-s, hepatitis, lymphography, and pima with several combinations of the regularization coefficients

| Problem | Generalization | | | | Number of nodes | | | |
|---|---|---|---|---|---|---|---|---|
| | (1,1) | (1,0) | (0,1) | (0,0) | (1,1) | (1,0) | (0,1) | (0,0) |
| Glass-g2 | 0.0475 | 0.0625 | 0.0675 | 0.0550 | 3.50 | 3.90 | 5.10 | 5.50 |
| Heart-c | 0.1145 | 0.1118 | 0.1066 | 0.1197 | 1.40 | 1.80 | 3.90 | 4.40 |
| Heart-s | 0.1221 | 0.1309 | 0.1529 | 0.1368 | 1.30 | 2.60 | 2.80 | 3.90 |
| Hepatitis | 0.0974 | 0.0947 | 0.0895 | 0.0947 | 1.00 | 1.00 | 1.20 | 1.10 |
| Lymphography | 0.1622 | 0.1622 | 0.1730 | 0.1432 | 1.70 | 1.90 | 1.60 | 5.10 |
| Pima | 0.1823 | 0.1922 | 0.1984 | 0.1995 | 1.00 | 1.00 | 1.90 | 3.60 |

The notation of regularization parameters is ($X$, $All$).

From these results we can affirm that the use of the regularization term is of interest, as the obtained networks are smaller and with a better generalization ability.

### 5.3. Effective number of hidden units

Understanding how and why an evolutive algorithm works is a very difficult issue. Nevertheless, any insight that we can gain on the way the algorithm performs is of great interest. In this section, we present some results that try to explain how the combinatorial crossover achieves its performance.

In a first step we studied the subspace spanned by the hidden nodes of the final network of each evolution. The network performs a non-linear projection of the input vector, $\mathbf{x}$, in a new subspace whose dimension is given by the number of hidden nodes, $h$, of the network. So every input pattern is transformed in a new point in the $h$-dimension subspace. In an attempt to study this new subspace we compute the principal components of the new point cloud generated by the hidden layer. The number of significantly non-zero components can be considered the *effective number of hidden units* (Weigend, 1993). Table 5 shows the ratio between effective number of hidden units and actual number of hidden units for the three methods in the problems studied. The table shows that the number of hidden units of the networks obtained by means of GaX y SaX is closer to the number of effective units. In this way, fewer unit are needed to classify the patterns, as each unit shares less information with the rest of the hidden units.

Table 5
Ratio between the effective number of hidden units and the actual number of hidden units for the 30 runs on every problem

| Problem | Effective/actual nodes ratio | | |
|---|---|---|---|
| | StdX | SaX | GaX |
| Anneal | 0.3221 | 0.4655 | 0.4828 |
| Autos | 0.3318 | 0.4143 | 0.3789 |
| Balance | 0.6774 | 0.5926 | 0.7500 |
| Cancer | 1.0000 | 1.0000 | 1.0000 |
| Card | 0.6286 | 1.0000 | 1.0000 |
| Gene | 0.3742 | 0.5000 | 0.4909 |
| German | 0.6136 | 1.0000 | 1.0000 |
| Glass | 0.3727 | 0.3771 | 0.3583 |
| Glass-g2 | 0.3213 | 0.4341 | 0.5213 |
| Heart-c | 0.5510 | 0.6786 | 0.8696 |
| Heart-s | 0.4000 | 0.6944 | 0.8400 |
| Hepatitis | 0.6667 | 0.8182 | 1.0000 |
| Horse | 0.3158 | 0.6250 | 0.6216 |
| Hypothyroid | 0.7600 | 1.0000 | 1.0000 |
| Ionosphere | 0.4462 | 0.9524 | 1.0000 |
| Iris | 0.4615 | 0.5000 | 0.5000 |
| Krvs.kp | 0.4902 | 1.0000 | 1.0000 |
| Labor | 0.7600 | 1.0000 | 1.0000 |
| Lymphography | 0.4038 | 0.5000 | 0.5000 |
| Pima | 0.4912 | 1.0000 | 1.0000 |
| Sick | 0.7813 | 1.0000 | 1.0000 |
| Sonar | 0.3067 | 0.3860 | 1.0000 |
| Vote | 1.0000 | 1.0000 | 1.0000 |
| Vowel | 0.3445 | 0.3571 | 0.3654 |
| Zoo | 0.3962 | 0.4167 | 0.4043 |

### 5.4. Functional representation of nodes

Our aim in this section is the visualization of the functionality of the nodes. We want to visualize how the nodes obtained by combinatorial optimization are more *different* among them. In order to show the functionality of a node, we follow the functional representation of a node of Moriarty and Miikkulainen (1997). In order to obtain the functional representation of a node, we calculate for each node its *function vector*. This function vector is made up of the outputs of the node when each input is set, by turn, to 1 and the rest to 0.

In order to represent the functionality of the node, we perform a principal component analysis of the function vectors, retaining the first two components. Following this method we can represent each node by a point in a two-dimensional space. The position of each node depends on its functionality, and we can consider that if two nodes are close in the two-dimensional space their functionality is somewhat similar. On the other hand, if two nodes are separated, their functionality must be very different. Fig. 7 shows the representation of the nodes of the best network of the first run of the algorithms for nine problems.

The figure shows how the nodes of the networks of the standard crossover are more clustered, with many nodes sharing similar functionality. On the other hand, the nodes of the combinatorial crossover tend to be more spread, so their collaboration is more effective. We can also see that in some networks evolved with standard crossover there are nodes grouped in pairs, these two nodes are probably the same node after some mutation. This effect does not appear when combinatorial crossover is used.

## 6. Conclusions

In this paper, we have proposed a new approach to neural network crossover. This approach partially avoids the disadvantages associated with the *permutation problem*. In this way, the main source of deception in the use of genetic algorithms for neural network evolution is removed.

We have performed an exhaustive comparison between the approach based on combinatorial optimization and the standard approach. This study has shown that the performance of the proposed model is better than the performance of the standard model. Moreover, the networks obtained with our model are more compact. Their size is among the smallest shown in the literature for solving the given problems.

Additional experiments have shown that the crossover based on this new approach significantly improves the performance of the evolutive process. On the other hand, the standard crossover has no significant effect over the performance of the evolution in several of the studied problems.

As a future research line we are working on the clustering of nodes before the crossover operation. Our idea is to group similar projections in a cluster before applying the combinatorial algorithm. Only a representative member of each cluster will be considered during the optimization.

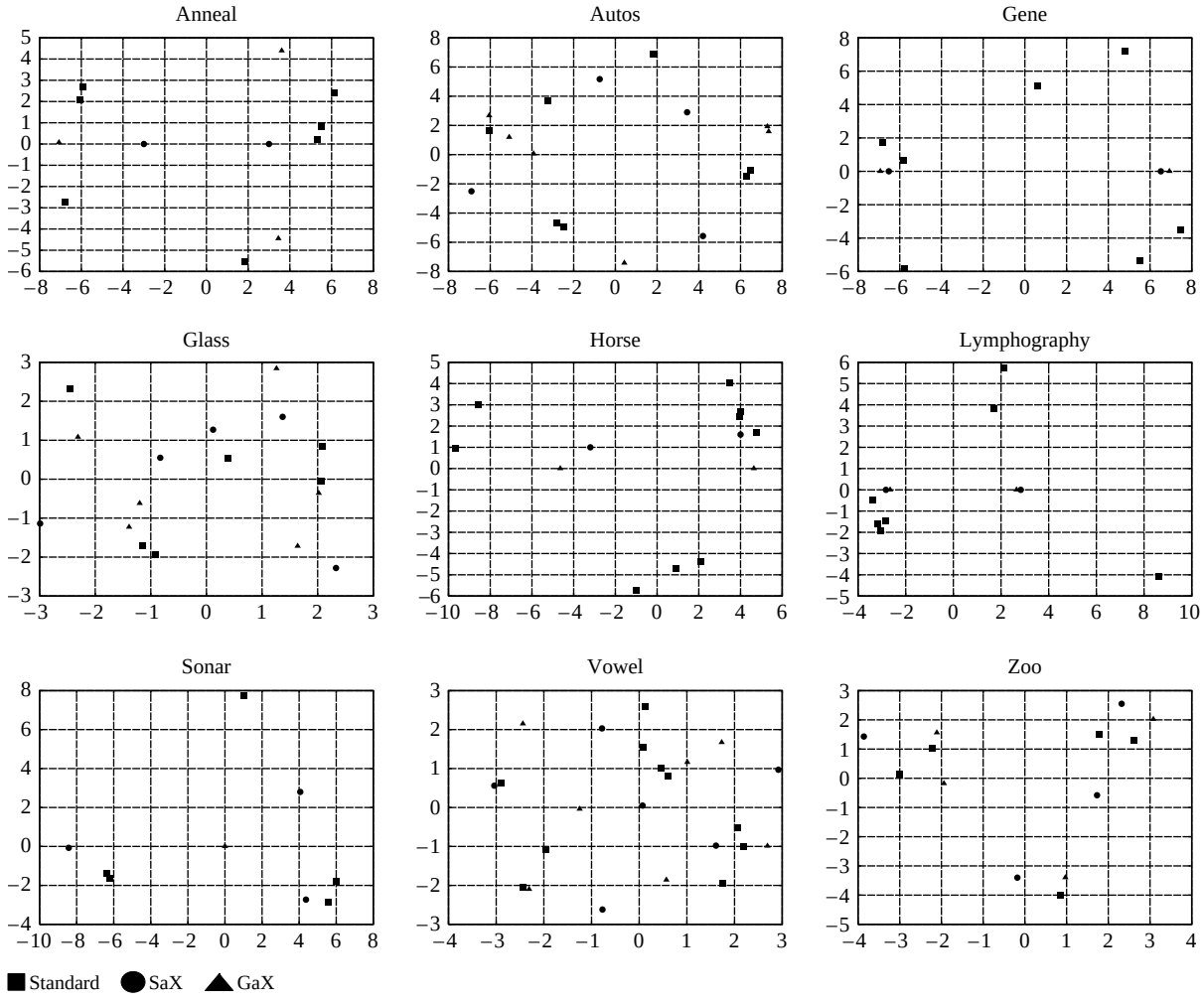


Fig. 7. Two first principal components for every network of the best ensemble of the first run for anneal, autos, gene, glass, horse, lymphography, sonar, vowel, and zoo problems.

## Acknowledgements

This work was supported in part by the Project TIC2002-04036-C05-02 of the Spanish CICYT and FEDER funds. The authors would like to acknowledge R. Moya-Sánchez for her helping in the final version of this paper and Dr F. Herrera-Triguero for his valuable advice.

## References

Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*(1), 54–65.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice*. New York, NY: Oxford University Press.

Balakrishnan, K., & Honavar, V. (1995). *Evolutionary design of neural architectures—a pleliminary taxonomy and guide to literature, technical report CS TR#95-01*. Artificial Intelligence Research Group, Department of Computer Science, Iowa State University, Ames, Iowa 50011-1040, USA (January 1995).

Branke, J. (1995). Evolutionary algorithms for neural network design and training. In J. Talander (Ed.), *Proceeding of the first nordic workshop on genetic algorithms and its applications, Vaasa, Finland, 1995*.

Cantú-Paz, E., & Kamath, C. (2003). Evolving neural networks to identify bent-double galaxies in the first survey. *Neural Networks*, *16*, 507–517.

Caruana, R., Lawrence, S., & Giles, L. (2001). Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems 13. Papers from neural information processing systems (NIPS) 2000* (pp. 402–408). Denver, CO: MIT Press.

Delahaye, D., & Odoni, A. (1997). Airspace congestion smoothing by stochastic optimization. In P. J. Angeline (Ed.), *Conference on evolutionary programming '97*. Indianapolis, IN: Springer.

Denison, D. G. T., Holmes, C. C., Mallick, B. K., & Smith, A. F. M. (2002). *Bayesian methods for nonlinear classification and regression. Wiley series in probability and statistics*. West Sussex, England: Wiley.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics—Part B*, *26*(1), 1–13.

Eshelman, L. J. (1990). *The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination*. San Mateo, CA: Morgan Kauffman.

Floreano, D., & Urzelai, J. (2000). Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, *13*, 431–443.

García-Pedrajas, N., Hervás-Martínez, C., & Muñoz-Pérez, J. (2002). Multi-objective cooperative coevolution of artificial neural networks. *Neural Networks*, *15*(10), 1255–1274.

García-Pedrajas, N., Hervás-Martínez, C., & Muñoz-Pérez, J. (2003). Covnet: A cooperative coevolutionary model for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, *14*(3), 575–596.

Goldberg, D. E. (1989a). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E. (1989b). Genetic algorithms and Walsh functions: Part 2, deception and its analysis. *Complex Systems*, *3*, 153–171.

Goldberg, D. E. (1989c). Genetic algorithms and Walsh functions: Part 1, a gentle introduction. *Complex Systems*, *3*, 129–152.

Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of Operations Research*, *13*, 311–329.

Hassibi, B., & Stork, D. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information systems* (Vol. 5).

Haykin, S. (1999). *Neural networks—A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall.

Hinton, G. E., & van Camp, D. (1993). Keeping neural networks simple by minimizing the description length of the weights. In *Sixth ACM international conference on computational learning theory ACM, Santa Cruz*, 1993 (pp. 5–13). New York, NY: ACM Press.

Karnin, E. D. (1990). A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, *1*(2), 239–242.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *IEEE international conference on neural networks*. Perth, Australia: IEEE Service Center.

Kirkpatrick, S., Gelatt Jr., C.D., & Vecchi, M.P. (1983). Optimization by simulated annealing Science, 220, 671–680.

Lawrence, S., Giles, C. L., & Tsoi, A. C. *What size neural network gives optimal generalization? Convergence properties of backpropagation, technical report.* Institute for Advanced Computer Studies, University of Maryland (August 1996).

Le Cun, Y., Denker, J. S., & Solla, S. A. (1990). Solla, optimal brain damage. In D. S. Touretzky (Ed.), *Advances in neural information processing (2)* (pp. 598–605). Denver, CO: Morgan Kaufmann.

Le Cun, Y. (1993). Efficient learning and second-order methods, a tutorial. In *Advances in neural information processing, Denver, CO* (Vol. 6).

Lerner, B., Guterman, H., Aladjem, M., & Dinstein, I. (1999). A comparative study of neural networks based feature extraction paradigms. *Pattern Recognition Letters*, *20*(1), 7–14.

Louis, S. J., & Li, G. (1997). Combining robot control strategies using genetic algorithms with memory. *Lecture Notes in Computer Science, Evolutionary Programming VI*, *1213*, 431–442.

Maniezzo, V. (1994). Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, *5*(1), 39–53.

Michalewicz, Z. (1994). *Genetic algorithms+data structures=evolution programs*. New York: Springer.

Miller, G. F., Todd, P. M., & Hedge, S. U. (1991). Designing neural networks. *Neural Networks*, *4*, 53–60.

Moriarty, D. E., & Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, *22*, 11–32.

Moriarty, D. E., & Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, *4*(5), 373–399.

Mozer, M. C., & Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky (Ed.), *Advances in neural information processing (1)* (pp. 107–155). Denver, CO: Morgan Kaufmann.

Odri, S. V., Petrovacki, D. P., & Krstonosic, G. A. (1993). Evolutional development of a multilevel neural network. *Neural Networks*, *6*, 583–595.

Pujol, J. C. F., & Poli, R. (1998). Evolving the topology and the weights of neural networks using a dual representation. *Applied Intelligence*, *8*(1), 73–84.

Schaffer, J. D., Whitley, L. D., & Eshelman, L. J. (1992). Combinations of genetic algorithms and neural networks: A survey of the state of the art. In L. D. Whitley, & J. D. Schaffer (Eds.), *Proceedings of COGANN-92 international workshop on combinations of genetic algorithms and neural networks* (pp. 1–37). Los Alamitos, CA: IEEE Computer Society Press.

Setiono, R. (1997). A penalty-function approach for pruning feedforward neural networks. *Neural Computation*, *9*, 185–204.

Setiono, R. (2001). Feedforward neural network construction using cross validation. *Neural Computation*, *13*, 2865–2877.

Smalz, R., & Conrad, M. (1994). Combining evolution with credit apportionment: A new learning algorithm for neural nets. *Neural Networks*, *7*(2), 341–351.

Thierens, D. (1996). Non-redundant genetic coding of neural networks. In *Proceeding of the 1996 IEEE international conference on evolutionary computation* (pp. 571–575). Piscataway, NJ: IEEE Press.

Thierens, D., Suykens, J., Vandewalle, J., & Moor, B. D. (1991). Genetic weight optimization of a feedforward neural network controller. In *Proceedings of the conference on neural nets and genetic algorithms* (pp. 658–663). Berlin: Springer.

Treadgold, N. K., & Gedeon, T. D. (1999). Exploring constructive cascade networks. *IEEE Transactions on Neural Networks*, *10*(6), 1335–1350.

van Rooij, A. J. F., Jain, L. C., & Johnson, R. P. (1996). *Neural networks training using genetic algorithms. Machine perception and artificial intelligence* (Vol. 26). Singapore: World Scientific.

Weigend, A. S. (1993). On overfitting and the effective number of hidden units. In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, & A. S. Weigend (Eds.), *Proceedings of the 1993 connectionist models summer school* (pp. 335–342). Hillsdale, NJ: Erlbaum Associates.

Weigend, A. S. (1994). On overfitting and the effective number of hidden units. In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elmand, & A. S. Weigend (Eds.), *Proceedings of the 1993 connectionist models summer school* (pp. 335–342). Hillside, NJ: Erlbaum Associates.

Whitehead, B. A., & Choate, T. D. (1996). Cooperative–competitive genetic evolution of radial basis function centers and widths for time series prediction. *IEEE Transactions on Neural Networks*, *7*(4), 869–880.

Yang, J., Parekh, R., & Honavar, V. DistAI: *An inter-pattern distance-based constructive learning algorithm, technical report TR #97-05.* Artificial Intelligence Research Group, Department of Computer Science, Iowa State University, Ames, Iowa (February 1997).

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, *9*(87), 1423–1447.

Yao, X., & Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, *8*(3), 694–713.