Evolutionary optimization of heterogeneous problems

Lluís A. Belanche Muñoz

Dept. de Llenguatges i Sistemes Informàtics.
Universitat Politècnica de Catalunya.
c/Jordi Girona Salgado 1-3 08034 Barcelona, Spain.
belanche@lsi.upc.es

Abstract. A large number of practical optimization problems involve elements of quite diverse nature, described as mixtures of qualitative and quantitative information, and whose description is possibly incomplete. In this work we present an extension of the *breeder genetic algorithm* that represents and manipulates this heterogeneous information in a natural way. The algorithm is illustrated in a set of optimization tasks involving the training of different kinds of neural networks. An extensive experimental study is presented in order to show the potential of the algorithm.

1 Introduction

Real-world data come from many different sources, described by mixtures of numeric and qualitative variables. These variables include continuous or discrete numerical processes, symbolic information, etc. In particular, qualitative variables might have a different nature. Some are *ordinal* in the usual statistical sense (i.e., with a discrete domain composed by k categories, but totally ordered w.r.t a given relation) or *nominal* (discrete but lacking an order). The data may also come with their own peculiarities (vagueness, uncertainty, incompleteness), and thus may require completely different treatments.

For Evolutionary Algorithms (EA), this translates in the introduction of a coding system (a suitable representation) for all the information not directly representable in the chosen chromosomic language. In practical terms, this involves the use of binary (or small cardinality) alphabets (as in classical genetic algorithms, GA) or real-valued quantities (as in Evolution Strategies, ES). This can be seen as a pre-processing that is not part of the original task, and as such may have deep consequences in the structure of the problem.

On the one hand, the algorithm receives an *indirect* and biased feedback (via the fitness function). One may argue that the decoding (even a complex one) can be seen as part of the genotype-to-phenotype development. While this posture makes sense in living organisms, in artificial systems data representation is a crucial factor for a successful learning process and can have a great impact on performance. The choice of representation should be as faithful as possible, in the sense that the relations between the represented entities should correspond to meaningful relations between the original data items. The possibility of injecting knowledge in the genetic operators about the kind of structures they are manipulating is also an appealing idea. Indeed, the choice of a representation scheme should not be made alone, but in conjunction with the genetic operators. A set of genetic operators that allows important partial solutions to propagate and does not permit invalid or unimportant ones to succeed is likely to enhance overall performance. For example, a good tree representation should be able to represent all possible trees (and only trees) and be "continuous" (i.e., similar representations should develop similar trees) [1]. Furthermore, genetic operators should generate only valid trees (or repair invalid trees after generation). In this way, higher-order gene interactions can be worked out in a controlled way.

On the other hand, unwanted gene interactions (epistatic phenomena) are one of the headaches in designing EA solutions. Problems with little or no epistasis are trivial to solve (e.g. by hillclimbing). However, highly epistatic problems may be extremely difficult to solve, since they can prevent the formation of useful building blocks, and are prone to deception. These interactions are very likely to appear as a side effect of the chosen coding scheme. Finally, results involving genes that encode information may be difficult to interpret in isolation, being the case that they do not represent any specific feature.

In this work, we extend the expressive power of the *breeder genetic algorithm* (BGA) [2] allowing it to manipulate heterogeneous information. We then apply it to a set of optimization tasks involving the training of a heterogeneous neural network. The results illustrate how performance and readability are enhanced.

2 The Breeder Genetic Algorithm

The Breeder Genetic Algorithm [2] is in midway between GAs and ESs. While in GA selection is stochastic and loosely inspired in natural selection, the BGA uses *truncation* selection, in which only the best individuals (usually a fixed percentage τ of the population size μ) are to be recombined and mutated. Genetic operators are applied by randomly picking two parents until the number of offspring equals $\mu - q$. Then, the former q best elements are re-inserted into the population, forming a new generation of μ individuals that replaces the previous one. The BGA selection mechanism is then deterministic (there are no probabilities), extinctive (the best elements are guaranteed to be selected and the worst are guaranteed *not* to be selected) and q-elitist (the best q elements always survive from a generation to the next). For the BGA, the typical value is $q = 1$. This is a form of the comma strategy (μ, λ) since the parents are not included in the replacement process, with the exception of the q previous best. Note that, given that q is fixed, only μ needs to be specified, since $\lambda = \mu - q$.

The BGA uses a *direct* representation, that is, a gene is a decision variable (not a way of coding it) and its allele is the value of the variable. An immediate consequence is that, in the absence of other conditionings as constraint handling, the fitness function equals the function to be optimized. In addition, the algorithm does not self-optimize any of its own parameters, as is done in ES and in some meta-GAs [3]. Chromosomes are thus potential solution vectors $\boldsymbol{x}$ of n components, where n is the problem size. This is of crucial importance since:

1. It eliminates the need of choosing a coding function for real numbers
2. It opens the way to the direct manipulation of different kinds of variables, other than real numbers (e.g., fuzzy, discrete) as *single* genes.
3. It permits the design of data-dependent genetic operators.

The BGA is mainly driven by recombination (very much as an ordinary GA), with mutation regarded as an important but background operator intended to reintroduce some of the alleles lost in the population. This view is conceptually right for GAs, because alphabet cardinality is usually very small (two, in most cases). However, for those algorithms that make use of real-valued alleles (like the BGA) mutation has to be seen in the double role of solution fine-tuner (for very small mutations) and as the main discovery force (for moderate ones). In fact, the initial BGA formulation remarked the *synergistic* effect of the combined and iterated application of recombination and mutation to extract the most from an EA [2]. We now briefly describe different possibilities for the genetic operators.

2.1 Recombination

Any operator mixing the genetic material of the parents is called a recombination operator. In a BGA, recombination is applied unconditionally. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ be two selected individuals $\boldsymbol{x}, \boldsymbol{y}$ such that $\boldsymbol{x} \neq \boldsymbol{y}$. Let $\boldsymbol{z} = (z_1, \ldots, z_n)$ be the result of recombination and $1 \leq i \leq n$. The following are some of the more common possibilities to obtain an offspring $\boldsymbol{z}$:

Discrete Recombination (DR). Set $z_i \in \{x_i, y_i\}$ (with equal probability).

Line Recombination (LR). Set $z_i = x_i + \alpha(y_i - x_i)$, with a fixed $\alpha \in [0, 1]$.

Extended Intermediate Recombination (EIR). Set $z_i = x_i + \alpha_i(y_i - x_i)$, with $\alpha_i \in [-\delta, 1 + \delta]$ chosen with uniform probability. The $\delta > 0$ parameter expresses the degree to which offspring can be generated out of the parents's scope, an imaginary line that joins them in $\mathbb{R}$. More precisely, $\delta|y_i - x_i|$ is the maximum fraction of the distance between parents where the offspring can be placed, either left to the leftmost parent or right to the rightmost parent. Reasonable values should not exceed $\delta = 0.5$, since the bigger the δ, the more the effect of the parents is diminished in creating offspring.

Fuzzy Recombination (FR). This operator replaces the uniform *pdf* (probability distribution function) by a bimodal one, with the two modes located at x_i, y_i. It thus favours offspring values close to either of the parents, and not in any intermediate point with equal probability, as with previous operators. The label "fuzzy" comes from the fact that the two parts of the *pdf* resemble fuzzy numbers (triangular in the original formulation [4]), fulfilling the conditions:

$$x_i - e|y_i - x_i| \leq t \leq x_i + e|y_i - x_i| \qquad y_i - e|y_i - x_i| \leq t \leq y_i + e|y_i - x_i|$$

stating that offspring t lies in one (or both) of the intervals, being $e > 0$ the fuzzy number's spread, the same for both parts. The favour for offspring values near the parents is thus stronger the closer the parents are. In the simplest case ($e = 0.5$) the two parts meet at the median and this point has zero probability.

2.2 Mutation

A mutation operator is applied to each gene with probability n^{-1} so that, on average, one gene is mutated for each individual. Let $\boldsymbol{z} = (z_1, \ldots, z_n)$ denote the result of mutation of an individual $\boldsymbol{x}$. The elements of $\boldsymbol{z}$ are formed as follows:

Discrete Mutation (DM). Set $z_i = x_i + sign \cdot range_i \cdot \delta$ with $sign \in \{-1, +1\}$ chosen with equal probability, $range_i = \rho(r_i^+ - r_i^-)$, $\rho \in [0.1, 0.5]$ and

$$\delta = \sum_{i=0}^{k-1} \varphi_i 2^{-i}$$

where $\varphi_i \in \{0, 1\}$ taken from a Bernouilli probability distribution such that $\Pr(\varphi_i = 1) = \frac{1}{k}$. In this setting $k \in \mathbb{N}^+$ is a parameter originally related to the *precision* with which the optimum was to be located. In practice, the value of k is related to the *expected* value of mutation steps: the higher k is, the more fine-grained the resultant mutation operator is. The factor ρ sets the *maximum* step that mutation is allowed to produce w.r.t. the range $[r_i^-, r_i^+]$ of variable i.

Continuous Mutation (CM). Same as DM with $\delta = 2^{-k\beta}$, where $\beta \in [0, 1]$ with uniform probability.

3 Extension of the BGA to Heterogeneous Problems

We consider basic data peculiarities most likely to be found in real applications. The BGA representation as well as the workings of the corresponding genetic operators are described. The algorithm manipulates the involved variables as a unique entity at all levels. Obviously, **real-valued** variables are directly treated as such, initialized at random within a pre-declared range, and recombined and mutated with the operators described in (§2.1) and (§2.2).

Ordinal m-valued variables are represented as positive natural numbers in the interval $[1, m]$ an initialized at random within the interval. For recombination, there are three possibilities, which mimic the real-valued operators: DR (generally valid but ignores the order), LR (respects the order), and EIR (idem, needs an δ parameter). Some preliminary investigations lead to the choice of LR ($\alpha = 0.5$), that is, the *median* of the parents. Mutation involves an *increase* (to the immediately following value w.r.t. the linear order) or a *decrease* (idem, but in the opposite sense), and the decision is taken with equal probability.

Nominal m-valued variables are also represented as an interval $[1, m]$, but no order relation is assumed. The clear choice for recombination is DR, being the only one explicitly assuming no underlying order. Mutation is realized by *switching* to a new value in the interval, with equal probability.

Fuzzy quantities. The extension to handle *fuzzy numbers* is given by a tuple of reals (three in the general case, two if the chosen representation is symmetric). *Linguistic variables* are described by their anchor points on the abscissa axis (e.g., four in the case of trapezoidal membership functions).

Recombination of fuzzy numbers is taken as the corresponding extension of the operators for real-valued quantities. In particular, for EIR the mode is obtained following its formula (involving the selection of δ), and the spread is computed using the formula with the *same* α. This makes sense since the spread is usually proportional to the mode. Fig. (1) provides an example.

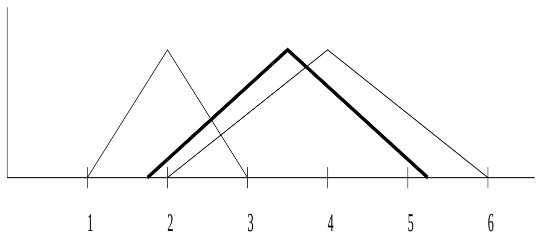


Fig. 1. EIR recombination for fuzzy numbers with $\delta = 0.25$, and $\alpha = 0.75$ uniformly chosen in $[-0.25, 1.25]$. Mode and spread for the two parents are 2.0, 1.0 and 4.0, 2.0. The thicker number is the result of recombination. As for real numbers, the value of α makes offspring resemble its bigger parent more (a factor of $\frac{3}{4}$) than its smaller one. The mode is 3.5 and the spread 1.75.

Mutation of fuzzy numbers is also developed as an extension of the real-valued operators, by taking into account that mode and spread are collectively expressing a single (fuzzy) number. Both continuous and discrete operators can be used, as follows. The change on the mode is determined using the respective formulas. The change on the spread uses the *same* sign and δ (which are the terms depending on probabilities) as used for the mode.

For *linguistic variables*, recombination is also an extension of the operators for real-valued quantities. For EIR, the procedure is analogous as for fuzzy numbers, that is, using the same α for all the involved quantities. In this case, however, the source of uncertainty is different, and there is no need for the offspring spreads to be in a proportion to their modes similar to that of the parents, and other operators could be conceivable. Fig. (2) provides an example. In mutation, a *single* step change is proposed according to the formulas, which affects all the constituting points (modes and spreads) in the same way.

Missing values are dealt with in a specially meaningful way. They are initially generated according to the estimated probability of a missing value in the vari-

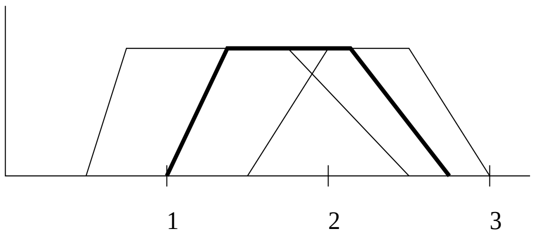


Fig. 2. EIR recombination for trapezoidal linguistic terms, with $\delta = 0.25$ and α uniformly chosen in $[-0.25, 1.25]$, and equal this time to 0.5, for clarity. The thicker set is the result of recombination. In this case, both parents are equally responsible of the obtained offspring ($\alpha = 0.5$).

able. This makes sense since for variables containing high numbers of missing values, the probability of placing one in the corresponding gene increases. If this probability is zero a missing value could still be introduced by mutation (signaling the temporal loss of a gene or trait). A mutation operator sets a missing value in the allele with a certain probability (usually very low). If this change leads to improved performance, it will be retained. A missing value cannot be mutated back to a non-missing one. A definite value can only be recovered by recombination to the (non-missing) gene of another individual.

Recombination is treated as *discrete* (DR) whenever at least one of the parents have a missing trait. This is coherent with the philosophy of EA: recombination stands for the *transmission* of the parents's genetic material to their offspring. If a parent is lacking a gene, this characteristic has to be given the chance to be passed on. Besides, if the trait or gene is lacking for both parents, it will be so for the offspring, since nothing can be "invented from scratch" (this is the role of mutation). In summary, given Ω a recombination operator (possibly heterogeneous), it is extended to a $\Omega_{\mathcal{X}}$ (where $\mathcal{X}$ denotes the missing value) as:

$$\Omega_{\mathcal{X}}(x_i, y_i) = \begin{cases} \Omega(x_i, y_i) & \text{if } x_i \neq \mathcal{X} \wedge y_i \neq \mathcal{X} \\ DR(x_i, y_i) & \text{if } x_i = \mathcal{X} \,\underline{\vee}\, y_i = \mathcal{X} \\ \mathcal{X} & \text{otherwise} \end{cases}$$

where $\underline{\vee}$ denotes exclusive-or. All this manipulation for missing values differs from the one that results by treating it as any other value, for its generation and propagation would be carried out blindly. The proposed treatment has the added appeal of being simple, and natural from the point of view of an EA in the sense that it is taken as a missing *gene* and is independent of the data type.

4 A Case Study in Neural Network Training

4.1 The BGA as a Neural Network Trainer

Evolutionary methods are immediate candidates for neural network optimization, being the case that they may alleviate the problem of local minima. However, when coding an ANN into a GA chromosome, highly complex interactions may develop, due to the influence that a given weight on a hidden unit has on the whole network computation. The usual binary representation of the real-valued weights carries with it extra interactions between non-neighbouring genes, thus inducing strong epistatic effects in GA processing. In these conditions, it is at least doubtful that the *building block hypothesis* can hold. A concise review on the generic use of evolutionary learning algorithms for neural optimization is given in [5, 6]. To the best of our knowledge, the BGA has only been used for some specific neural optimization tasks or application examples [7, 8].

4.2 Heterogeneous Neural Networks

These artificial networks are built out of neuron models defined as a mapping $h : \hat{\mathcal{H}}^n \rightarrow \mathbb{R}$. Here $\hat{\mathcal{H}}^n$ is a cartesian product of an arbitrary number n of *source sets*. These source sets may be extended reals $\hat{\mathcal{R}}_i = \mathbb{R}_i \cup \{\mathcal{X}\}$, extended families of (normalized) fuzzy sets $\hat{\mathcal{F}}_i = \mathcal{F}_i \cup \{\mathcal{X}\}$, and extended finite sets of the form $\hat{\mathcal{O}}_i = \mathcal{O}_i \cup \{\mathcal{X}\}$, $\hat{\mathcal{M}}_i = \mathcal{M}_i \cup \{\mathcal{X}\}$, where each of the $\mathcal{O}_i$ has a full order relation, while the $\mathcal{M}_i$ have not. The special symbol $\mathcal{X}$ extends the source sets and denotes the unknown element (missing information), behaving as an *incomparable* element w.r.t. any ordering relation. According to this definition, neuron inputs are vectors composed of n elements among which there might be reals, fuzzy sets, ordinals, categorical and missing data [11].

An heterogeneous neuron computes a *similarity index*, followed by a classical squashing non-linear function with domain in $[0, 1]$. We use in this work a *Gower-like* similarity index [9] in which the computation for heterogeneous entities is constructed as a weighed combination of partial similarities over single variables. This coefficient has its values in the real interval $[0, 1]$ and for any two objects i, j given by tuples of cardinality n, is given by the expression:

$$s_{ij} = \frac{\sum_{k=1}^{n} g_{ijk}\, \delta_{ijk}}{\sum_{k=1}^{n} \delta_{ijk}}$$

where g_{ijk} is a similarity *score* for objects i, j according to their value for variable k. These scores are in the interval $[0, 1]$ and are computed according to different schemes for numeric and qualitative variables. In particular, for a continuous variable k and any two objects i, j the following similarity score is used:

$$g_{ijk} = \hat{s}\left(\frac{|v_{ik} - v_{jk}|}{\sup_{i,j} |v_{ik} - v_{jk}|}\right)$$

Here, v_{ik} denotes the value of object i for variable k and $\hat{s}(z) = (1 - z)^{\alpha}, \alpha > 0$. The similarity measure used for categorical variables is the *overlap*:

$$g_{ijk} = \begin{cases} 1 & \text{if } v_{ik} = v_{jk} \\ 0 & \text{if } v_{ik} \neq v_{jk} \end{cases}$$

The δ_{ijk} is a binary function expressing whether both objects are *comparable* or not according to their values w.r.t. variable k, as follows:

$$\delta_{ijk} = \begin{cases} 1 & v_{ik} \neq \mathcal{X} \text{ and } v_{jk} \neq \mathcal{X} \\ 0 & \text{otherwise} \end{cases}$$

For variables representing fuzzy sets, if $\mathcal{F}_i$ is an arbitrary family of fuzzy sets in X, and $\tilde{A}$, $\tilde{B} \in \mathcal{F}_i$, the following similarity relation s is used:

$$s(\tilde{A}, \tilde{B}) = \sup_{u \in X} \{min\ (\mu_{\tilde{A}}(u), \mu_{\tilde{B}}(u))\}$$

Notice that this measure is reflexive in the strong sense and also symmetric. As for the activation function, a modified version of the classical logistic is used [10], which is an automorphism (a monotonic bijection) in $[0, 1]$.

The framework has provision for other types of variables, as ordinal or linguistic, and other kinds of combination for the partial similarities. The resulting heterogeneous neuron is sensitive to the *degree of similarity* between its weight vector and a given input, both composed in general by a mixture of continuous and discrete quantities –possibly with missing data– and can be used for configuring heterogeneous artificial neural networks (HNN).

5 An Experimental Study

Seven learning tasks taken from the Proben repository [12] are studied, altogether representative of the kinds of variables typically found in real problems, while displaying different degrees of missing information (from 0% to 26%). Their main characteristics are displayed in Table 1.

For every data set, the available documentation is analysed in what concerns type and meaning of variables, allowing an assessment on the more appropriate treatment. Missing information is also identified. Specifically, some originally "continuous" variables are treated as ordinal since this makes much more sense than to consider them as continuous. Examples would be *number of times pregnant* or *heart pulse*. There are also variables that, besides being endowed with a total order relation, display a source of vagueness (coming from their *subjective* character) that has to be modeled. This is the case of, for instance, the *temperature of extremities* (`cold, cool, normal, warm`) or the *abdominal distension* (`none, slight, moderate, severe`). These variables are treated as linguistic by respecting the number and order of the (initially crisp) values. In absence of more precise information, the cut points are set at the 0.5 level, as is usually done. Besides, some continuous variables are converted to triangular fuzzy numbers with a low fuzziness (roughly estimated at a 0.5%) reflecting the uncertainty derived from their imprecise measurements.

Table 1. Basic features of the data sets: Def. (default accuracy), Missing (percentage of missing values), Missing-1 (percentage of cases with at least one missing value).

Name	Type	#Cases	Def.	Missing	Missing-1	In→Out	Data
Pima Diabetes	C	768	65.1%	10.6%	48.8%	$8 \rightarrow 2$	6R, 0N, 2I
Credit Card	C	690	55.5%	0.65%	5.4%	$15 \rightarrow 2$	6R, 9N, 0I
Horse Colic	C	364	61.5%	26.1%	98.1%	$20 \rightarrow 3$	5R, 5N,10I
Heart Disease	C	920	55.3%	16.2%	67.5%	$13 \rightarrow 2$	3R, 6N, 4I
Solar Flares	R	1066	-	0.0%	0.0%	$9 \rightarrow 3$	0R, 5N, 4I
Sinus-Cosinus	R	400	-	0.0%	0.0%	$2 \rightarrow 1$	2R, 0N, 0I
SISO-Bench	R	500	-	0.0%	0.0%	$2 \rightarrow 1$	2R, 0N, 0I

(C classification R regression) (R real N nominal I ordinal)

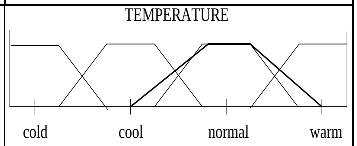
Four different architectures are studied, composed of a hidden layer of 4, 8, 12 and 16 neurons plus as many output units as required by the task. The data sets are split in 5 folds, and ten 60%-20%-20% partitions are formed, such that each of the 5 folds appears exactly twice as a validation fold, twice as a test, and 6 times as one of the 3 training folds. For each configuration, ten runs are carried out, varying the initial conditions of the BGA, set to $\mu = 100, \tau = 25$. The task is to minimize the *normalized root square error* (NRSE) on the training part, until $30,000$ error evaluations are used in each run. Test errors are computed using the network that produced the lowest error in its validation part.

As a reference, two standard neural networks are also trained in the same conditions, treating all information as real-valued quantities, as is the case in standard neural learning systems. Specifically, the standard scalar-product neuron plus a bias weight, using a logistic as activation function (MLP) and a radial basis function neuron (RBF) based on Euclidean distance, plus a Gaussian with its own variance (to be learned). The BGA is here used in its original formulation. The obtained *generalization results* are summarized in Table 2, where mean test NRSE is displayed, averaged over the four architectures.

Table 2. Results for each data set, averaged over the four architectures. An asterisk (*) means that the result is worse than any of the other two, in the sense that for *all* of the four architectures, Mann-Whitney tests for "greater than" are significant at the 95% level (w.r.t. *both* of the other two models). The average is given as an indication.

Problem	MLP net	RBF net	HNN net
Pima Diabetes	0.692	0.774 (*)	0.700
Horse Colic	0.926(*)	0.714	0.725
Heart Disease	0.661(*)	0.586	0.581
Credit Card	0.677	0.740 (*)	0.533
Solar Flares	1.182(*)	0.855	0.937
Sinus-Cosinus	0.083	0.384 (*)	0.042
SISO-Bench	0.022	0.115 (*)	0.023
Average	0.606	0.595	0.506

Table 3. Heterogeneous weights of a hidden neuron for the *Horse Colic* problem.

#	Name	Type	Value
2	Rectal temperature (celsius)	*fuzzy number*	37.3 ± 2.1
5	Temperature of extremities	*linguistic*	
7	Mucous membranes color	*nominal*	"normal pink"
14	Nasogastric reflux PH	*fuzzy number*	2.1 ± 0.3
19	Abdominocentesis appearance	*nominal*	"cloudy"

The *readability* of the obtained solutions is illustrated in Table 3. We show part of the weights of a hidden neuron taken at random from one of the networks delivered by cross-validation for the *Horse Colic* problem. Linguistic terms are shown in graphical form, whereas triangular fuzzy numbers are shown in numerical form (rounded to one decimal) for clarity. Notice the obtained linguistic term is almost symmetric, a characteristic found by the algorithm itself.

Acknowledgements: This work is supported by the Spanish CICYT TAP99-0747.

References

1. Palmer, C.C., Kershenbaum, A. Representing trees in genetic algorithms. In Bäck, Th., Fogel D.B., Michalewicz, Z. (Eds.) *Handbook of Evolutionary Computation*. IOP Publishing & Oxford Univ. Press, 1997.
2. Mühlenbein, H., Schlierkamp-Voosen, D. Predictive Models for the Breeder Genetic Algorithm. *Evolutionary Computation*, 1 (1): 25-49, 1993.
3. Bäck, Th. *Evolutionary Algorithms in Theory and Practice*. Oxford Press, 1996.
4. Voigt, H.M., Mühlenbein, H., Cvetkovic, D. Fuzzy recombination for the continuous Breeder Genetic Algorithm. In Procs. of ICGA'95.
5. Balakrishnan, K., Honavar, V. Evolutionary design of neural architectures - a preliminary taxonomy and guide to literature. Technical report CS-TR-95-01. Dept. of Computer Science. Iowa State Univ., 1995.
6. Yao, X. Evolving Artificial Neural Networks. Procs. of the IEEE, 87(9), 1999.
7. De Falco, I., Iazzetta, A, Natale, P., Tarantino, E. Evolutionary Neural Networks for Nonlinear Dynamics Modeling. In Procs. of PPSN V, Amsterdam, 1998.
8. Zhang, B.T., Mühlenbein, H. Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor. *Complex Systems*, 7(3): 199-220, 1993.
9. Gower, J.C. A General Coefficient of Similarity and some of its Properties. *Biometrics*, 27: 857-871, 1971.
10. Valdés J.J., Belanche, Ll., Alquézar, R. Fuzzy Heterogeneous Neurons for Imprecise Classification Problems. *Intl. Journal of Intelligent Systems*, 15(3): 265-276, 2000.
11. Belanche, Ll. Heterogeneous neural networks: theory and applications. Ph.D. Thesis. Universitat Politècnica de Catalunya, Barcelona, Spain, 2000.
12. Prechelt, L. Proben1: A set of Neural Network Benchmark Problems and Benchmarking Rules. Facultät für Informatik. Univ. Karlsruhe. Tech. Rep. 21/94, 1994.