Zio Enrico
Department of Nuclear Engineering, Polytechnic of Milan, Italy

Soft computing methods applied to condition monitoring and fault diagnosis for maintenance

Keywords

soft computing, artificial neural networks, fuzzy logic, genetic algorithms, condition monitoring, fault diagnosis, maintenance

Abstract

Malfunctions in equipment and components are often sources of reduced productivity and increased maintenance costs in various industrial applications. For this reason, machine condition monitoring is being pursued to recognize incipient faults in the strive towards optimising maintenance and productivity. In this respect, the following lecture notes provide the basic concepts underlying some methodologies of soft computing, namely neural networks, fuzzy logic systems and genetic algorithms, which offer great potential for application to condition monitoring and fault diagnosis for maintenance optimisation. The exposition is purposely kept on a somewhat intuitive basis: the interested reader can refer to the copious literature for further technical details.

1. Introduction

Managing an industrial plant entails evaluating and trading off the conflicting objectives of economic service and safe operation. The first scientific approaches to this management problem date back to the 1950's and 1960's and can be found in the review paper by McCall [1] and in the book from Barlow and Proschan [3]. As a result, various so-called periodic maintenance *optimisation models* were introduced in which both costs and benefits of *periodic maintenance* were quantified and an optimum compromise between the two was sought. Well-known models originating from this period are the so-called age and *block replacement* models.

From the practical point of view, at that time, preventive maintenance was strongly advocated as a means to reduce failures, for safety reasons, and unplanned downtime, for economic reasons. In many companies, large time-based preventive maintenance programs were set up.

Nowadays, modern production plants are expected to run continuously for extended hours. In this situation, unexpected downtime due to components and equipment failures has become more costly than ever before. The faults can degrade the quality of a product line or even cause the entire plant to function incorrectly, possibly resulting in downtime of the production system with consequent economic loss.

On the other hand, proper monitoring of the conditions of the components and systems can be highly cost effective in minimizing maintenance downtime by providing advanced warning and lead time to prepare the appropriate corrective actions upon an adequate fault diagnosis.

For this reason, condition monitoring has become a popular approach for predicting component failures using physical information on the actual state of the equipment. The possibility of monitoring the system state, continuously for operating systems or by tests and inspections for stand-by safety systems, allows a more dynamic preventive maintenance practice, called condition-based maintenance (CBM), in which the decision of maintaining the system is taken on the basis of the observed condition of the system. This, in principle, allows saving resources by preventively maintaining the system only when necessary. In many practical instances, this approach has proved more effective than the previous large preventive maintenance programs.

Analytical results for single-component deteriorating systems have been established under simplifying assumptions. Markov and semi-Markov models have been the preferred approach in modelling CBM [4], [12], [16]- [17], [21], [27] but other approaches, like counting processes [1], have also been proposed. The majority of the models appeared in the literature

assume that the system's degradation level can only be known through periodic inspection as typical in safety systems such as those employed in nuclear plants [2]-[3], [26]-[27]; Kopnov [16] considers the case in which the system is continuously monitored and Lam [17] considers both cases. Another common assumption is to consider that repairs/replacements always restore the system to a 'good-as-new' condition, which, in practice, may not be very realistic; Kopnov [16] has allowed also for partial recovery.

The dynamic CBM policies for single-component systems whose condition can only be known through inspection, developed in [10], [12] and [19], are all based on control-limit rules which define when to repair/replace a component and when to schedule the next inspection.

For the continuously inspected systems investigated by Kopnov [16], the two-level policies from the Inventory Theory have been adapted to the CBM problem of degrading systems. Semi-Markov processes are also considered; a death process is proposed for a unit subject to corrosion and a Markov chain is used for modelling fatigue crack growth.

A common feature of the models discussed is that the state of the system is described as a state of a Markov process and then the analysis proceeds to finding analytically the probabilities of the various states. However, if the system is made of several multi-state components the analysis becomes excessively complicated. Simulation tools are hence needed when treating more complex systems. Bérenguer et al. [[4] have extended the work of Grall et al. [10] by investigating two-component deteriorating systems using simulation. Their maintenance model takes into consideration economic dependence between components and again the state of the system is only known through periodic inspections. Barata et al. [2] developed a stochastic degradation model for repairable multi-component systems and embedded its simulation within a maintenance optimisation scheme. The condition of each component is known continuously. The novelty of the model stems from the fact that the component's failures can occur not only because of excessive degradation which leads to a critical state of the system, but also because of random shocks which suddenly fail the system and whose occurrence probability is degradation-dependent. While in some cases the system degradation level depends on the combination of many mechanisms and can only be known through inspection [4], [12], [12], [19], [26] there are other mechanisms such as fatigue and corrosion of structures in which deterministic laws are known and the uncertainty is on the value of the parameters that govern those laws.

Regarding the deterioration models themselves, Hontelez et al. [12] give several examples, all of deterministic nature, from the civil engineering field. Grall et al. [10] use a model in which the degradation level increases randomly according to an exponential distribution. Degradation models describing fatigue and corrosion of metal structures are described by Guedes Soares and Garbatov in [23], [24].

The success of condition monitoring and condition-based maintenance strongly relies on the capability of modelling the degradation processes and the corresponding plant dynamic responses under different configurations and conditions. However, the complexity and non-linearities of the involved processes are such that analytical modelling becomes burdensome, if at all feasible without resorting to unrealistic simplifying assumptions. For this reason, empirical modelling is becoming very popular since it does not require a detailed physical understanding of the processes nor knowledge of the material properties, geometry and other characteristics of the plant and its components and it does not resort to simplifying assumptions: the underlying dynamic model is identified by fitting plant operational data, with a procedure often referred to as 'learning' or 'training'.

Among the various techniques of empirical modelling, the so-called soft computing methods offer powerful algorithms for constructing non-linear models from operational data. As a fact, they are being used with increasing frequency as an alternative to traditional models in a variety of engineering applications including monitoring, prediction, diagnostics, control and safety.

The main soft computing methodologies are Neural Networks (NNs), Fuzzy Logic Systems (FLSs) and Genetic Algorithms (GAs). These methodologies are inspired by biology and natural behaviour and provide potentially powerful tools for effectively tackling difficult multivariate, non-linear problems, which often cannot be solved with ease by means of traditional analytical or numerical methods.

In the present lecture notes, we shall try to give a brief description of the concepts underlying the different methodologies and point out their main advantages and limitations. With this objective in mind, we shall refer our discussion to a multidimensional non-linear input/output mapping, for NNs and FLSs, or searching space, for GAs optimisation.

NNs and FLSs are capable of establishing the existing non-linear input/output relationships, which map the inputs of a system to its outputs. They reconstruct the complex non-linear relations by combining multiple simple functions. More precisely, through an analogy with the functioning of the human brain, NNs form the shape of the mapping of interest by appropriately combining a large number of sigmoid, radial or other

simple parameterised functions, which are adjusted (enlarged, shrunk, shifted, etc.) by means of appropriate parameters and synaptic weights [20], [21]. The great power of this technique lies in the fact that the adjustments can be made 'automatically' through a training phase based on available input/output data: this training phase allows to adjust the NN-model parameters so as to obtain the best interpolation of the multivariate, non-linear functional relation between input and output.

FLSs, on the contrary, partition the input/output spaces into several typically overlapping areas, whose shapes are established by assigned membership functions and whose mapping relationships are governed by distinct, simple IF-THEN rules [28], [13]. The great advantage of this method lies in the inherent capability of handling imprecise data and in the physical transparency and interpretability offered by this particular way of representing the underlying model relations.

Finally, if the input/output multidimensional space is seen as a searching space in which the inputs are the decision variables and the outputs are the performance indicators of the search problem, the GAs offer a powerful method for evaluating a best input solution with respect to the optimisation (minimization or maximization) of the performance indicators of interest [9], [11]. The main advantages of the method are that the search is performed by manipulation of a population of points, contrary to classical methods which proceed from a single solution point to another, and that the search is solely based on the evaluation of the performance indicators, with no need of other information, e.g. of derivative nature.

2. Artificial Neural Networks

Artificial neural networks (ANNs) are information processing systems composed of simple processing elements (nodes) linked by weighted connections. Their functioning is inspired by the biological neural networks.

A biological neuron consists of dendrites, a cell body and axons (*Figure 1a*). The connections between a dendrite and the axons of other neurons are called synapses. In correspondence of each synapse, electric pulses from other neurons are transformed into chemical information which is input to the cell body: if the sum of the inputs received by the neuron through all its synapses exceeds a given threshold, then it fires an electric pulse which activates the neuron function. The network of all these neurons makes up the most essential part of the human brain and its operation enables the incredible variety of human activities. In synthesis, the function of a biological neuron is 'simply' to output pulses, with the characteristics of a quasi-step switching function, according to a weighed combination of the multiple signals received from the other connected neurons. A second important function of the neuron is to appropriately modify the rate of transition through the different synapses to optimise the whole network.

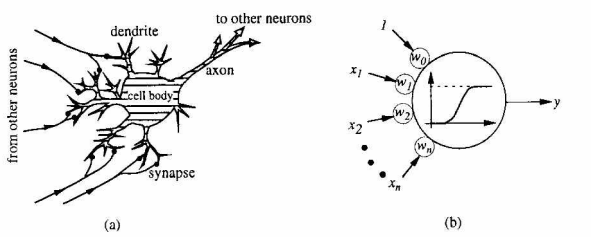

Figure 1. A biological neuron (a) and an artificial neuron model (b) [25]

An artificial neuron (node) aims at simulating the operation of a biological neuron: thus, it accepts multiple inputs $x_1, x_2, ..., x_m$, it weighs them by means of adaptive synaptic weights, $w_0, w_1, ..., w_m$, and it simulates the switching function characteristic of the input/output relation to provide the output (*Figure 1b*). Connecting several artificial neurons together one obtains an artificial neural network which, by construction, constitutes an information processing system composed of simple processing elements (nodes) linked by weighted synaptic connections [21]. The adaptation of the synaptic weights is realized through a training phase during which properly devised learning algorithms are used to change the synaptic weights of the network in an effort to optimise its mapping performance [20].

Here, we limit ourselves to briefly describing the most commonly used multi-layered feed-forward neural network which, in its simplest form, consists of three layers of processing elements: the input, the hidden and the output layers, with n_i, n_h and n_o nodes, respectively (*Figure 2*). The signal is processed forward from the input to the output layer. Each node collects the output values, weighted by the connection weights, from all the nodes of the preceding layer, processes this information through a sigmoid function

$$f(x) = \left(1 + e^{-x}\right)^{-1}$$

and then delivers the result towards all the nodes of the successive layer. Typically, both input and hidden layers are provided with an additional bias node, which serves as a threshold in the argument of the activation function and whose output always equals unity.

As for the determination of the connection weights, i.e. the model parameters, the learning technique most commonly employed is the so-called error back-propagation algorithm, which follows from the

general gradient-descent method [20]. In short, starting from random values of the synaptic weights the back-propagation algorithm performs the steepest descent in the weight space on a surface whose height at any point is equal to the error function; in practice, it consists of an iterative gradient algorithm designed to minimize the mean square error between the actual network output and the true value. A number n_p of sets (patterns) of input and associated outputs are repeatedly presented to the network and the values of the connection weights are modified so as to minimize the average squared output deviation error function, or Energy function, defined as:

$$E = \frac{1}{2n_p n_o} \sum_{n=1}^{n_p} \sum_{l=1}^{n_o} (\hat{y}_{nl} - y_{nl})^2 \qquad (1)$$

where y_{nl} and $\hat{y}_{nl}$ are the true output value of the *n*-th pattern and the corresponding network-computed output value at the *l*-th node, $l = 1, 2, ..., n_o$. Through this training procedure, the network is able to build an internal representation of the input/output mapping of the problem under investigation. The success of the training strongly depends on the normalization of the data and on the choice of the training parameters. Typically, each signal is transformed by an affined mapping in an interval such as (0.2, 0.8) or similar and the connection weights are initialised randomly within an interval such as (-0.3, 0.3) or similar.

After the training is completed, the final connection weights are kept fixed. New input patterns are presented to the network, which is capable of recalling the information stored in the connection weights during training to produce the corresponding output, coherent with the internal representation of the input/output mapping. Notice that the non-linearity of the sigmoid function of the processing elements allows the neural network to learn arbitrary non-linear mappings [6], [15]. Moreover, each node acts independently of all the others and its functioning relies only on the local information provided through the adjoining connections. In other words, the functioning of one node does not depend on the states of those other nodes to which it is not connected. This allows for efficient distributed representation and parallel processing, and for an intrinsic fault-tolerance and generalization capability.

These attributes render the artificial neural networks a powerful tool for signal processing, non-linear mappings and near-optimal solution to combinatorial optimisation problems.

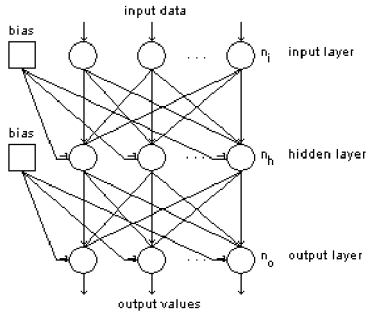

Figure 2. Scheme of a three-layered, feedforward neural network

2.1. Feedforward artificial neural networks for regression

In order to understand further the way of functioning of neural networks, let us consider an artificial feedforward neural network to be trained for performing the task of non-linear regression, i.e. estimating the underlying non-linear relationship existing between a multi-dimensional vector of input variables *x* and an output target *y*, assumed mono-dimensional for simplicity of illustration ($n_0 = 1$, in esq. (1)), based on a finite set of input/output data examples (the above mentioned patterns),

$$D \equiv \left\{ (x_n, y_n),\ n = 1, 2, ..., n_p \right\}.$$

It is assumed that the target *y* is related to the input vector *x* by an unknown deterministic function $\mu_y(x)$ corrupted by a *white noise* ε, viz.

$$y = \mu_y(x) + \varepsilon(x);\ \varepsilon(x) \quad N(0, \sigma_\varepsilon^2(x)) \qquad (2)$$

The objective of the regression task is to estimate $\mu_y(x)$ by means of a *regression function* $f(x; \hat{w})$, dependent on a set of parameters $\hat{w}$ to be properly determined on the basis of an available set of input/output patterns *D*.

A feedforward neural network provides a non-linear form of the function $f(x; \hat{w})$ for the regression task. As above explained, the parameters $\hat{w}$ are called network weights and are usually determined by a *training* procedure which aims at minimizing the *quadratic error function*

$$E = \frac{1}{2n_p} \sum_{n=1}^{n_p} (\hat{y}_n - y_n)^2 \qquad (3)$$

where for simplicity of notation the output node subscript *l* has been dropped since the case considered concerns a single output.

The network output corresponding to input x_n is a function of the weight values, $y_n = f(x;w)$. If the network architecture and training parameters are suitably chosen and the minimization done to determine the weights values is successful, the obtained function $f(x;w)$ gives a good estimate of the unknown, true regression function $\mu_y(x)$. Indeed, it is possible to show that in the ideal case of an infinite training data set and perfect minimization algorithm, a neural network trained to minimize the error function in (3) provides a function f which performs a mapping from the input x into the expected value of the target y conditioned on x, i.e. the true deterministic function $E[y|x] = \mu_y(x)$ [5]. In other words, the network averages over the noise on the data and discovers the underlying deterministic generator. Unfortunately, all the training sets are finite and there is no guarantee that the selected minimization algorithm can achieve the global minimum.

The quadratic error function in (3) can be motivated from the principle of maximum likelihood applied to the set of available training patterns $D \equiv \{(x_n, y_n),\ n = 1, 2, ..., n_p\}$. The likelihood of the observed data set D is defined as

$$L(w|D) = \prod_{n=1}^{n_p} p(x_n, y_n|w) = \prod_{n=1}^{n_p} p(y_n|x_n, w)\, p(x_n|w) \tag{4}$$

where it is assumed that each pattern (x_n, y_n) is drawn independently from the same distribution $P(x_n, y_n)$. The unknown weights w of the neural model $f(x;w)$ are determined by maximization of the likelihood $L(w|D)$ of observing the training set D [5]. Instead of maximizing the likelihood, it is computationally more convenient to minimize its negative logarithm,

$$L^*(w|D) = -\ln L(w|D) = -\sum_{n=1}^{n_p} \ln p(y_n|x_n, w) - \sum_{n=1}^{n_p} \ln p(x_n|w) \tag{5}$$

The distribution of the input values x_n, i.e. the second term in the rhs of (5), is independent of the parameters w of the neural model $f(x;w)$; thus, the parameters w can be found by minimization of the first term only, i.e. the following error function

$$E = -\sum_{n=1}^{n_p} \ln p(y_n|x_n, w) \tag{6}$$

Different forms of the conditional distribution $p(y|x,w)$ lead to different error functions. In particular, the assumption of a Gaussian distribution for the target as in (2) leads to a quadratic error function of the kind in (3) [20].

Indeed, from eq. (2) we have

$$\varepsilon(x) = y - \mu_y(x) \sim N\left(0, \sigma_\varepsilon^2(x)\right)$$

and using the regression function $y = f(x;w)$ to estimate $\mu_y(x)$, we get

$$p(y|x,w) = \frac{1}{\sqrt{2\pi}\sigma_\varepsilon(x)} e^{-\frac{1}{2}\frac{(y-y)^2}{\sigma_\varepsilon^2(x)}} \tag{7}$$

and the error function in (6) becomes

$$E = -\sum_{n=1}^{n_p} \ln\left[\frac{1}{\sqrt{2\pi}\sigma_\varepsilon(x_n)} e^{-\frac{1}{2}\frac{(y_n - y_n)^2}{\sigma_\varepsilon^2(x_n)}}\right]. \tag{8}$$

When the noise variance is independent of the input x, i.e. $\sigma_\varepsilon^2(x) = \sigma_\varepsilon^2$,

$$E = \frac{1}{2\sigma_\varepsilon^2}\sum_{n=1}^{n_p}(y_n - y_n)^2 + n_p \ln\sigma_\varepsilon + \frac{n_p}{2}\ln 2\pi, \tag{9}$$

and the error function reduces to the form (3) since the other terms do not depend on the weights w of the neural model.

Obviously, the quadratic error function in (3) can be used also for regression on targets, which are not Gaussian-distributed: in this case, the resulting regression function $f(x;w)$ cannot distinguish between the true distribution and any other with same mean and variance.

Finally, notice that the value of the error function (3) at the minimum gives a measure of the variance of the target data, averaged over the input.

2.2. Neural network uncertainty

In practical regression problems, there are two types of prediction that one may want to obtain in correspondence of a given input x: an estimate

$f(x;\mathbf{w})$ of the underlying deterministic function $\mu_y(x)$ and an estimate of the target value *y* itself, as given by eq.(2), with their corresponding measures of confidence. This requires that the various sources of uncertainty affecting the determination of the weights $\mathbf{w}$ be properly accounted for [7].

For what concerns the estimate $f(x;\mathbf{w})$ of $\mu_y(x)$, it must be considered that, from a probabilistic point of view, the data set $D \equiv \{(x_n, y_n),\ n = 1,2,...,n_p\}$ used for training the network is only one of an infinite number of possible data sets. This variability in the training data set is due to the variability in the sampling of the input vectors $x_n, n = 1,2,...,n_p$ and in the random fluctuation of the corresponding target output y_n. Each possible training set *D* can give rise to a different set of network weights $\mathbf{w}$. Correspondingly, there is a distribution of regression functions $f(x;\mathbf{w})$ with variance (with respect to the training set *D*):

$$E\left\{\left[f(x,\mathbf{w}) - E[f(x,\mathbf{w})]\right]^2\right\} \tag{10}$$

Since in practice a neural network structure is not a perfect algorithm, it systematically under/over estimates the correct result, i.e. the expected value $E[f(x,\mathbf{w})]$ is not equal to the true underlying deterministic function, $\mu_y(x)$, their difference being the so-called *bias*. Of course, the bias would be zero in the case of a perfect neural network.

To quantify the confidence in the estimate $f(x;\mathbf{w})$ of the true deterministic function $\mu_y(x)$, it is customary to refer to the *confidence intervals* of the error $f(x,\mathbf{w}) - \mu_y(x)$ whose variance with respect to all possible training data sets is:

$$E\{[f(x,\mathbf{w}) - \mu_y(x)]^2\}$$

$$= E\left\{\left[\begin{array}{l} f(x,\mathbf{w}) - E[f(x,\mathbf{w})] + \\ \\ + E[f(x,\mathbf{w})] - \mu_y(x) \end{array}\right]^2\right\}$$

$$= \left\{\begin{array}{l} \left[f(x,\mathbf{w}) - E[f(x,\mathbf{w})]\right]^2 \\ \\ + \left[E[f(x,\mathbf{w})] - \mu_y(x)\right]^2 \\ \\ + 2\left[f(x,\mathbf{w}) - E[f(x,\mathbf{w})]\right] \\ \\ \cdot \left[E[f(x,\mathbf{w})] - \mu_y(x)\right] \end{array}\right\}$$

$$= \left\{\left[f(x,\mathbf{w}) - E[f(x,\mathbf{w})]\right]^2\right\}$$

$$+ \left\{E[f(x,\mathbf{w})] - \mu_y(x)\right\}^2 \tag{11}$$

where the first term is the variance of the distribution of regression function values $f(x;\mathbf{w})$ and measures the extent to which the network regression function $f(x;\mathbf{w})$ differs from the ensemble average over different training data sets, whereas the second term is the square of the bias which measures the extent to which the average (over all possible training data sets) of the network regression function $f(x;\mathbf{w})$ differs from the true underlying deterministic function, $\mu_y(x)$.

2.3. Sources of uncertainty

A first source of uncertainty comes from a wrong choice of the network architecture. Indeed, in case of a network with too few nodes, i.e. too few parameters, a large bias occurs since the regression function $f(x;\mathbf{w})$ has insufficient flexibility to model the data adequately, which results in poor generalization properties of the network. On the other side, excessively increasing the flexibility of the model by introducing too many parameters increases the variance term because the network regression function tends to over-fit the training data. Thus, in both cases, the network performs poorly when fed with new input pattern in the generalization phase. A trade-off is, then, necessary. This trade-off is typically achieved by controlling the model complexity (i.e., the number of parameters) and the training procedure (by adding a regularization term in the error function or by early stopping of the training [5]) so as to achieve a good fit of the training data but with a reasonably smooth regression function which is not over-fit to the data.

An additional source of uncertainty in the network performance, due to uncertainty in the weights $\mathbf{w}$, arises from the minimization algorithm itself, which may get stuck in a local minimum of the error function. Furthermore, the training may be stopped prematurely, before reaching the minimum.

Besides the above uncertainties in the regression function $f(x;\mathbf{w})$ due to uncertainty in the weights $\mathbf{w}$, in practice there is also uncertainty in the input values *x* due to noise and uncertainty in the model structure $f(\cdot)$ itself.

For what concerns the prediction of the target value, *y*, it is clear that even in the ideal case of a regression function $f(x;\mathbf{w})$ equal to the true deterministic function, $\mu_y(x)$, the target *y* could not be predicted

with certainty due to the presence of the noise term $\varepsilon(x)$ in (eq. 2) which accounts for the intrinsic random fluctuations. To quantify the accuracy of the estimate of *y*, it is customary to refer to the *prediction intervals* of the deviation

$$y - f(x, \hat{w}) = \left[\mu_y(x) - f(x, \hat{w})\right] + \varepsilon \ .$$

The variance of such deviation is:

$$\begin{aligned} & E\left\{\left[f(x, \hat{w}) - y\right]^2\right\} \\ & = E\left\{\left[f(x, \hat{w}) - \mu_y(x)\right]^2\right\} + E\left\{\left[\varepsilon\right]^2\right\} \qquad (12) \\ & = E\left\{\left[f(x, \hat{w}) - \mu_y(x)\right]^2\right\} + \sigma_\varepsilon^2 \end{aligned}$$

Note that the first term is the variance of the distribution of the error $f(x, \hat{w}) - \mu_y(x)$ (eq. 11), so that the prediction intervals include the confidence intervals.

From the above said, it appears that artificial neural networks are *unstable* predictors: small changes in the training data may produce very different regression models and consequently different generalization performances on new, unseen data. For this reason, the generalization performance of a single artificial neural network, particularly if trained on small data sets, should be tested by means of a *k*-fold cross validation, where the available set of n_p input/output patterns is divided into *k* subsets of (approximately) equal size and the network is trained *k* times on a training set in which each time one of the *k* subsets is left out and used to verify the network generalization performance. If *k* equals the sample size, the procedure is called *leave-one-out* cross-validation [14].

3. Fuzzy logic system

Fuzzy logic systems are founded on the theory of fuzzy sets, which, in general, deals with vague information, where *vagueness* is defined as the uncertainty associated with linguistic or intuitive information. For example, the quality of an image may be judged as *bad*, *medium* or *good*. From this example, it appears that vagueness is related to immeasurable issues and involves situations in which the transitions among linguistic statements occur across boundaries, which are not sharp.

A few words on the concepts underlying fuzzy set theory seem then in order [28]. Let us consider a variable *x*, for example a measured output of a plant. Mutating from classical logic, the set *U* which contains all the possible values of *x* is usually called the universe of discourse (UOD) of *x* or the universal set. Suppose that the UOD *U* has been subdivided in a sequence of subsets $X_i \subset U$. In classical set theory, the X_i's are mutually exclusives so that a given value of *x* may belong to only one of them. These sets are called *crisp* and the membership of a crisp value of *x* to a set X_i is specified by the (rectangular) characteristic function χ_{X_i}, which is equal to unity or zero according to whether the value of *x* belongs or not to X_i.

In fuzzy set theory, the situation is quite different: the subsets X_i of the universe of discourse *U* of a linguistic variable *x* are not necessarily exclusive, so that a given crisp value of $x \in U$, may simultaneously belong to more than one of them with different degrees of membership. This feature clearly distinguishes fuzzy set theory from probability theory, which operates on crisp events. In fuzzy set theory, then, the subsets are not identified by sharp boundaries but by linguistic labels (*words*). For example we may consider the linguistic variable *temperature* defined in the universe of discourse $U = (0°, 40°)$ subdivided in the subsets $X_1 = (0°, 20°)$, $X_2 = (10°, 30°)$, $X_3 = (20°, 40°)$, labelled by the words *cold*, *warm* and *hot*, respectively. Clearly a given temperature value may belong to more than one set, e.g. 15° belongs to X_1 and X_2.

Fuzzy set theory aims at quantifying the meanings of the words attached by the analyst to the subsets X_i (such as cold, warm or hot in the above example) within the framework of set theory. To this aim, to each set X_i the analyst associates, for all values of $x \in U$, the membership function $\mu_{X_i}(x)$, which represents the degree to which he postulates that *x* belongs to X_i. As opposed to the characteristic functions of classical set theory, which are rectangular in shape and disjoint, the membership functions associated to fuzzy sets have subjective shapes and may overlap to describe a continuous transition from one set to another, thus providing for the possibility that a given value of $x \in U$ simultaneously belongs to several sets with different degrees of membership.

In summary, in the fuzzy context we deal with linguistic variables (e.g. *temperature*) whose arguments are words, also called fuzzy values (e.g. *negative*, *approximately zero*, *low*, *positive*, *high*). Each of these words refers to a subset of the universe of discourse and the degree of membership of the crisp values within the subset is analytically specified by the associated membership function.

Fuzzy logic systems build on the theory of fuzzy sets to realize a complex non-linear input/output relation as a synthesis of multiple simple input/output relations. This idea is similar to that of NNs. The difference is that in FLSs each simple input/output relation is embedded in a different IF-THEN rule with 'fuzzy', and not sharp, boundaries so that going from one rule to the other the system output gradually changes [28].
In general, the generic *j*-th fuzzy rule is made up of a number of antecedent and consequent linguistic statements, suitably related by fuzzy connections:

IF (x_1 is X_{1j}) AND (…) AND (x_m is X_{mj})
THEN (y_1 is Y_{1j}) AND (…) AND (y_k is Y_{kj})

The linguistic variables $x_p, p = 1, 2, ..., m$, are the antecedents, represented in terms of the fuzzy sets X_{pj} of the universe of discourse (range) X_p, with membership functions $\mu_{X_{pj}}(x_p)$. The linguistic variables $y_q, q = 1, 2, ..., k$, are the consequents, represented by the fuzzy sets Y_{qj} of the universe of discourse Y_q, with membership functions $\mu_{Y_{qj}}(y_q)$.
The connective operator AND links two fuzzy concepts and it is generally implemented by means of a *t-norm*, typically the minimum operator $\wedge$ or the algebraic product. Since one of the key features of FLS lies in allowing the overlapping of the rules, a given input vector (FACT) will typically activate more than one rule.
Another feature of FLSs is the ability to separate logic and fuzziness [25]. Conventional binary logic systems are unable to do so and thus their governing rules have to be modified when either the system logic or the variables fuzziness needs to be changed. On the contrary, FLSs modify their rules when the logic needs to be changed whereas they modify the supporting membership functions when fuzziness should be changed. To clarify this, consider the performance of an inverted pendulum controller [25]. Define as ω and ω' the angle that the pole forms on the right side with the vertical line and the associated angular velocity, respectively. Let two correct logic rules for the control of the pendulum be:

1) IF ω is positive big AND ω' is big, THEN move the car to the right quickly
2) IF ω is negative small AND ω' is small, THEN move the car to the left slowly

If the performance of the controller is unsatisfactory, one needs not change the fuzzy rules themselves, which are logically correct, but rather must only modify appropriately the definition of fuzziness in the linguistic terms *big*, *small*, *quickly*, *slowly*.
On the other hand, binary logic rules such as

3)IF $40° < \omega < 60°$ AND $50° s^{-1} < \omega' < 80° s^{-1}$, THEN move the car at $0.5 m/s$
4)IF $-20° < \omega < -10°$ AND $10° s^{-1} < \omega' < 20° s^{-1}$, THEN move the car at $-0.1 m/s$

must be modified whenever the logic of the system or the quantitative definitions of angle, angular velocity and car speed are changed.
To understand FLSs, we address on an intuitive basis the problem of controlling a plant [25]. The mathematical-based approach of classical and modern control theory stems on the observation of the system, the construction of its mathematical model and the design of a model-based controller. The focus is placed on the behaviour of the target system and on its mathematical representation.
On the contrary, fuzzy-logic control does not utilize the target system for modelling but it is based, in principle, on the linguistic control rules used by experienced and skilled operators. Although most skilled operators do not know the mathematical behaviour of the systems they are required to control, they can still perform successfully. For example, a skilled driver most likely ignores the mathematical equations underlying the physical behaviour of the car when turning to the right while driving up an unpaved hill and, yet, he or she can still handle the car safely and successfully. In this view, a fuzzy logic controller aims at reproducing the knowledge and experience supporting the control actions of skilled human operators using IF-THEN fuzzy rules.
Clearly the set of IF-THEN fuzzy rules constitutes the heart of the input/output mapping model provided by the FLS. When the experience of the skilled human operators is unavailable or insufficient, because of the complexity of the system, input/output data can be used to generate a set of fuzzy rules representative of the mapping from the input space into the output one. This phase of rule construction during which both the system input and output are known is often referred to with the term 'training', in analogy to the procedure for determining the weights of a neural network model illustrated in Section 2.

3.1. Establishing the antecedent part of a rule

The IF part of a rule is called *antecedent*. Establishing the antecedent parts of the rules of the FLSs is related to the partitioning of the multivariate input space. For simplicity, let us consider a two-dimensional input space (x_1, x_2) and a one-dimensional output space *y*.

Most of the times it is possible to assume that all input variables are independent and, thus, partition separately the input space in each direction (*Figure 3*). This assumption makes it easy not only to partition the input space but also to interpret the partitioned areas in linguistic terms. For example, the rule IF *temperature* is A_1 AND *humidity* is A_2, THEN …, is easy to understand because the variables of temperature and humidity are separated. The difference between 'crisp' and 'fuzzy' rule-based systems lies in the way the input space is partitioned (*Figure 3*). The idea behind FLSs is that in the real analog world, changes are not sudden and sharp but gradual in nature so that overlapping of rules domains should be allowed. The degree of overlapping is defined in terms of membership functions and the intrinsic gradual property allows for smooth control.

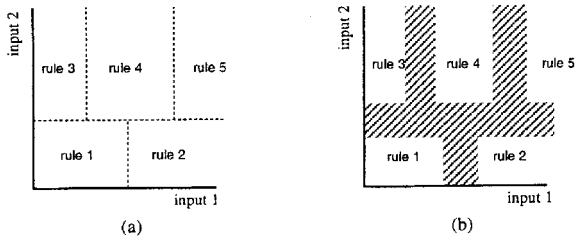

Figure 3. Rule partition of an input space (a) partition for crisp rules and (b) partition for fuzzy rules [25]

3.2. Establishing the consequent part of a rule

The THEN part of a rule is called *consequent*. In the control case, establishing the consequent parts of the rules of the FLSs must eventually lead to defining the control action value corresponding to each rule. In this respect, fuzzy models are classified into three types according to the form used for the consequent *y*:

Model type	Consequent expression	Characteristic
Mamdani	*Y is Y*	*Y* is a fuzzy set
Takagi Sugeno Kang (TSK)	$y = c_0+\Sigma_i a_i x_i$	a_i 's are constant and the x_i 's are the input variables
Simplified fuzzy	$y = c$	*c* is constant

In the Mamdani type FLSs the consequent is a fuzzy variable defined by a membership function. These systems are more difficult to compute than those whose consequents are numerically defined but they better describe the qualitative knowledge related to the consequent.
The consequents of the TSK models are expressed as a weighed linear combination of the input variables. It is also possible to use non-linear combination for better performance, but at the expense of the transparency of the rules.
The simplified fuzzy model has fuzzy rules whose consequents are constant values. Thus, it is a special case of both Mamdani and TSK types. Even if the output of each rule is a constant, the overall FLS output is non-linear because it contains the characteristics of the underlying model membership functions.

3.3. Inferring the output corresponding to a given input: fuzzy reasoning and aggregation

Now that the IF and THEN parts of the rules have been designed, the next step is to infer the output of the FLS resulting from a given *m*-dimensional crisp input vector $x^{(*)}$, also called FACT. This is done in two steps: 1) determination of the rules strengths; 2) aggregation of each rule output into the final FLS numerical output.
As mentioned, the connective operator AND links two fuzzy concepts in a rule and it is generally implemented by means of a *t-norm* applied to the membership functions of the rule's antecedents evaluated in correspondence of the crisp input vectors $x^{(*)}$ constituting the FACT. Typically, the minimum operator $\wedge$ or the algebraic product is employed. This gives the strength of the rule for the given FACT: a measure of how active that rule is for the given FACT or, in other words, how much the FACT is described by the antecedents of the rule. Considering a generic rule *l*, the strength $s_l(x^{(*)})$ is given, in the case of the algebraic product, by the product of its antecedent membership values in correspondence of the crisp inputs:

$$s_l(x^{(*)}) = \prod_{p=1}^{m} \mu_{X_{pl}}(x_p^{(*)}) \qquad (13)$$

where X_{pl} denotes the fuzzy set characterizing the *p*-th antecedent of the *l*-th rule.
In the case of the min operator:

$$s_l(x^{(*)}) = \min_{\mu}(\mu_{X_{pl}}(x_p^{(*)})) \qquad (14)$$

Obviously, if the *l*-th rule is not activated by $x^{(*)}$ then its strength is zero. This occurs if at least one of the elements of the crisp input vector, say $x_p^{(*)}$, does not belong to X_{pl}, the corresponding fuzzy set in the rule.
Let us now consider the aggregation step and denote by R_f the number of rules, which are activated (fired)

by the input $x^{(*)}$ and by y^l the consequent in the *l*-th activated rule, $l = 1, 2, ..., R_f$. The output *y* can be determined as a normalized weighed sum of the consequents y^l of the R_f active rules, the weights being the strengths of the rules,

$$y = \frac{\sum_{l=1}^{R_f} s_l(\overset{\circ}{x}^{(*)}) y^l}{\sum_{l=1}^{R_f} s_l(\overset{\circ}{x}^{(*)})} \tag{15}$$

Figure 4 shows an example of a simple TSK-type FLS with four fuzzy rules [25]. The first rule for example could be:

IF x_1 is small AND x_2 is small,
THEN $y = 3x_1 + 2x_2 - 4$

Corresponding to the input vector $(x_1, x_2) = (10, 0.5)$, the membership functions of the fuzzy sets constituting the antecedents of the first rule are readily evaluated as 0.8 and 0.3.
If the algebra product is used as the *t*-norm operator for the AND connection, then the rule strength is $s_1(10, 0.5) = 0.8 \cdot 0.3 = 0.24$. Similarly, the strengths of the second, third and fourth rules are $s_2(10, 0.5) = 0.8$, $s_3(10, 0.5) = 1.0$, $s_4(10, 0.5) = 0.3$, respectively. The output of each rule corresponding to the given input vector is $y^1 = 27$, $y^2 = 23.5$, $y^3 = -9$, $y^4 = -20.5$, respectively. Then, the final system output is

$$y = \frac{0.24 \cdot 27 + 0.8 \cdot 23.5 + 1 \cdot (-9) + 0.3 \cdot (-20.5)}{0.24 + 0.8 + 1.0 + 0.3}$$

$$\approx 4.33 \tag{16}$$

In the Mamdani type model, with fuzzy consequents, the output of the fuzzy inference engine consists of a fuzzy set $Y' \subseteq Y$ with compact support (η_1, η_2), whose membership function is $\mu_{Y'}(y)$. However, eventually we are interested in finding a crisp number y^* that represents the information encoded in the output fuzzy set Y'. This conversion, called *defuzzification*, may be done in several ways, the most commonly used being the *Centre of Area* (COA) method:

$$y^* = y_{COA} = \frac{\int_{\eta_1}^{\eta_2} y \cdot \mu_{Y'}(y) dy}{\int_{\eta_1}^{\eta_2} \mu_{Y'}(y) dy} \tag{17}$$

The crisp number y^* thereby obtained can be taken as the output resulting from the given input vector (FACT) $x^{(*)}$.

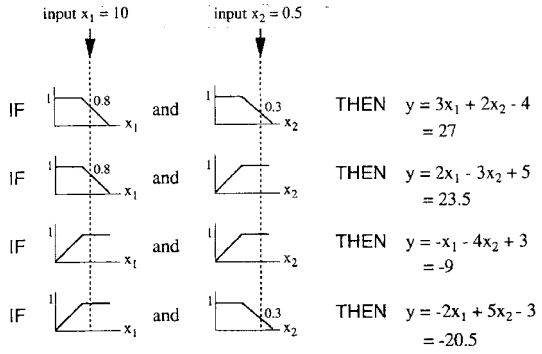

Figure 4. Example aggregation of TSK model [25]

3.4. Interpretation of fuzzy rules

The major difference between fuzzy systems and other non-linear approximates, such as neural networks, is the possibility of interpretation of the rules underlying the fuzzy model. This, however, does not automatically follow from the existence of a rule structure. Therefore, it is relevant to discuss the circumstances under which a fuzzy system is really interpretable and this depends on the application. Yet, some general guidelines can be given. The following factors may influence the interpretability of a fuzzy logic system [25]:

- *Number of rules*. If the number of rules is too large, the fuzzy system can be hardly interpreted. Especially for systems with many inputs, the number of rules often becomes overwhelmingly large if all antecedent combinations are realized.
- *Number of antecedents in the rule premise*. Rules with premises that have many, say more than three or four, antecedents are hard to interpret. In the human language, most rules include only very few antecedents even if the total number of inputs relevant for the problem is large.
- *Dimension of input fuzzy sets*. One-way to avoid, or at least, reduce the difficulties with high-dimensional input spaces and to decrease the number of rules is to resort to multi-dimensional input fuzzy sets. However, multidimensional input fuzzy sets with more than three inputs are certainly beyond human imagination and it is precisely the conjunction of one-dimensional

input fuzzy sets that make a fuzzy logic model interpretable.

- *Arrangement of fuzzy sets*. Fuzzy sets should be arranged in proper order over the universe of discourse so that, for example, very small is followed by *small*, followed by medium, large and so on. If the fuzzy model is developed on the basis of expert knowledge such ordering comes natural. However, if rule construction by training on input/output data is used to optimise the FLS, the ordering of the fuzzy sets might be lost if no precautions are taken. This typically leads to constraint optimisations in which for example the ordering of the input membership functions is constrained. This not only leads to an easier interpretation of the fuzzy system but, often, also provides an improved performance.

4. Genetic algorithms

Search or optimisation algorithms inspired on the biological laws of genetics are called evolutionary computing algorithms [8]. The main features of these algorithms are that the search is conducted i) using a population of multiple solution points or candidates, ii) using operations inspired by the evolution of species, such as breeding and genetic mutation, iii) based on probabilistic operations, iv) using only information on the objective or search function and not on its derivatives. Typical paradigms belonging to the class of evolutionary computing are genetic algorithms (GAs), evolution strategies (ESs), evolutionary programming (EP) and genetic programming (GP). In the following we shall focus on the more popular GAs.

As a first definition, it may be said that genetic algorithms are numerical search tools aiming at finding the global maximum (or minimum) of a given real objective function of one or more real variables, possibly subject to various linear or non linear constraints [11]. Genetic algorithms have proven to be very powerful search and optimisation tools especially when only little about the underlying structure in the data is known. They employ operations similar to those of natural genetics to guide their path through the search space. Essentially, they embed a survival of the fittest optimisation strategy within a structured, yet randomised, information exchange [9].

Since the GAs owe their name to the fact that their functioning is inspired by the rules of the natural selection, the adopted language contains many terms borrowed from biology, which need to be suitably redefined to fit the algorithmic context. Thus, when we say that the GA operates on a set of (artificial) chromosomes, these must be understood as strings of numbers, generally sequences of binary digits 0 and 1. If the objective function has many arguments, each string is partitioned in as many substrings of assigned lengths, one for each argument and, correspondingly, we say that each chromosome is analogously partitioned in (artificial) genes. The genes constitute the so-called genotype of the chromosome and the substrings, when decoded in real numbers called control factors, constitute its phenotype. When the objective function is evaluated in correspondence of the values of the control factors of a chromosome, its value is called the fitness of that chromosome. Thus each chromosome gives rise to a trial solution to the problem.

The GA search is performed by constructing a sequence of populations of chromosomes, the individuals of each population being the children of those of the previous population and the parents of those of the successive population. The initial population is generated by randomly sampling the bits of all the strings. At each step, the new population is then obtained by manipulating the strings of the old population in order to arrive at a new population hopefully characterized by an increased mean fitness. This sequence continues until a termination criterion is reached. As for the natural selection, the string manipulation consists in selecting and mating pairs of chromosomes in order to groom chromosomes of the next population. This is done by repeatedly performing on the strings the four fundamental operations of reproduction, crossover, replacement and mutation, all based on random sampling. These operations will be detailed below [18].

Finally, it is by now acknowledged that GAs take a more global view of the search space than many other optimisation methods. The main advantages are i) fast convergence to near global optimum, ii) superior global searching capability in complicated search spaces, iii) applicability even when gradient information is not readily achievable. The first two advantages are related to the population-based searching property *(Figure 5*). Indeed, while the gradient method determines the next searching point using the gradient information at the current searching point, the GA determines the next set of multiple search points using the evaluation of the objective function at the current multiple searching points. When only gradient information is used, the next searching point is strongly influenced by the local geometric information of the current searching point so that the search may remain trapped in a local minimum. On the contrary, the GA determines the next multiple searching points using the fitness values of the current searching points, which are spread throughout the searching space, and it can also resort to the additional mutation to escape from local minima.

The key disadvantage of a GA is that its convergence speed becomes slow near the global optimum.

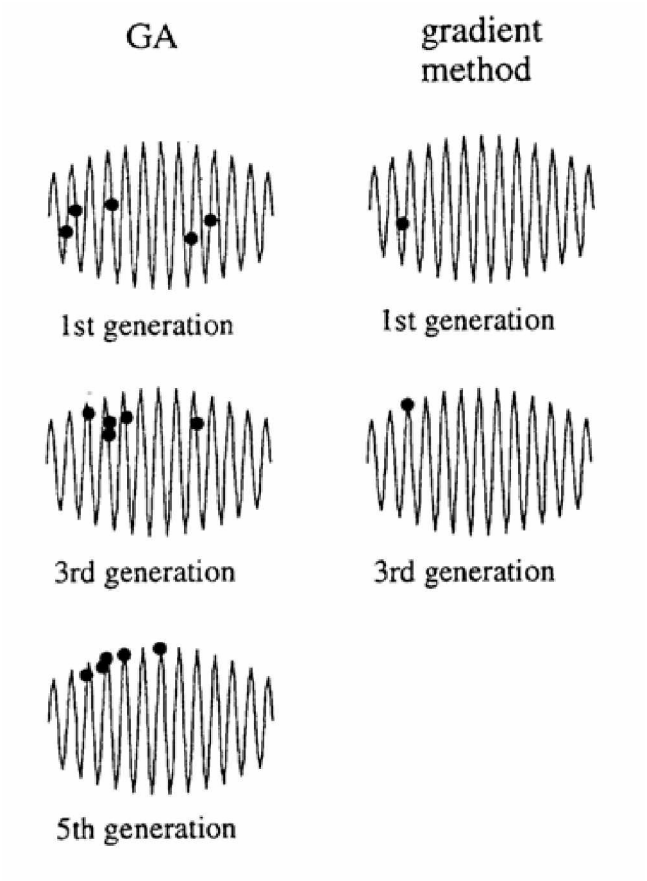

Figure 5. GA search and gradient-based search [25]

4.1. Definitions

It is important to be acquainted with the technical terms of GAs.

Individuals and Population: An individual is a chromosome, constituted by $n \geq 1$ genes and a population is a collection of individuals. To code/decode the i-th gene in a control factor, that is in an argument of the objective function, the user:

- defines the range (a_i, b_i) of the corresponding argument in the objective function;
- assigns the resolution of that independent variable by dividing the range (a_i, b_i) in 2^{n_i} intervals. A number n_i of bits is then assigned to the substring representative of the gene and the relation between a real value $x \in (a_i, b_i)$ and its binary counterpart β is

$$x = a_i + \beta \frac{b_i - a_i}{2^{n_i}} \qquad (18)$$

The values a_i, b_i, n_i are called the *phenotyping parameters* of the gene.

Figure 6 shows the constituents of a chromosome made up of three genes and the relation between the genotype and the external environment, i.e. the phenotype, constituted by three control factors, x_1, x_2, x_3, one for each gene. The passage from the genotype to the phenotype and vice versa is ruled by the phenotyping parameters of all genes, which perform the coding/decoding actions. Each individual is characterized by fitness, defined as the value of the objective function calculated in correspondence of the control factors pertaining to that individual. Thus a population is a collection of points in the solution space, i.e. in the space of *f*.

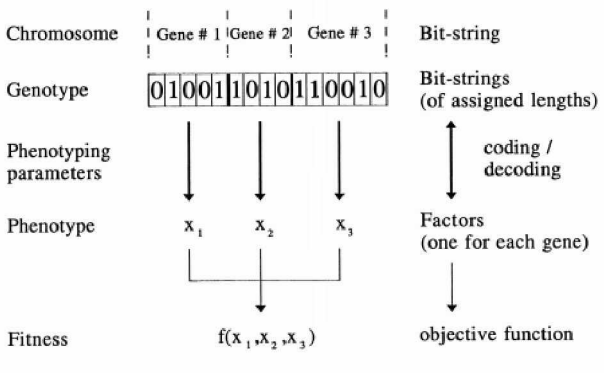

Figure 6. Components of an individual (a chromosome) and its fitness

An important feature of a population is its genetic diversity: if the population is too small, the scarcity of genetic diversity may result in a population dominated by almost equal chromosomes and then, after decoding the genes and evaluating the objective function, in the quick convergence towards an optimum which may well be a local one. At the other extreme, in too large populations, the overabundance of genetic diversity can lead to clustering of individuals around different local optima: then the mating of individuals belonging to different clusters can produce children (newborn strings) lacking the good genetic part of either of the parents. In addition, the manipulation of large populations may be excessively expensive in terms of computer time.

In most computer codes the population size is kept fixed at a value set by the user so as to suit the requirements of the model at hand. The individuals are left unordered, but an index is sorted according to their fitnesses. During the search, the fitnesses of the newborn individuals are computed and the fitness index is continuously updated.

4.2. Creation of the initial population

As said above, the initial population is generated by random sampling the bits of all the strings. This procedure corresponds to uniformly sampling each control factor within its range. The chromosome creation, while quite simple in principle, presents

some subtleties worth to mention: indeed it may happen that the admissible hypervolume of the control factors is only a small portion of that resulting from the Cartesian product of the ranges of the single variables, so that one must try to reduce the search space by resorting to some additional condition in terms of suitable physical criteria to be satisfied. This remark also applies to the *chromosome replacement*, below described.

4.3. The traditional breeding algorithm

The breeding algorithm is the way in which the $(n+1)$-th population is generated from the *n*-th previous one.

The first step of the breeding procedure is the generation of a temporary new population. Assume that the user has chosen a population of size N (generally an even number). The population reproduction is performed by resorting to the Standard Roulette Selection rule: to find the new population, the cumulative sum of the fitnesses of the individuals in the old population is computed and normalized to sum to unity. The new population is generated by random sampling individuals, one at a time with replacement, from this cumulative sum, which then plays the role of a cumulative distribution function (cdf) of a discrete random variable (the position of an individual in the population). By so doing, on the average, the individuals in the new population are present in proportion to their relative fitness in the old population. Since individuals with relatively larger fitness have more chance to be sampled, most probably the mean fitness of the new population is larger.

The second step of the breeding procedure, i.e. the *crossover*, is performed as indicated in *Figure 7*: after having generated the new (temporary) population as above said, *N/2* pairs of individuals, the parents, are sampled at random without replacement and irrespectively of their fitness, which has already been taken into account in the first step. In each pair, the corresponding genes are divided into two portions by inserting at random a separator in the same position in both genes (one-site crossover): finally, the first portions of the genes are exchanged. The two chromosomes so produced, the children, are thus a combination of the genetic features of their parents. A variation of this procedure consists in performing the crossover with an assigned probability p_c (generally rather high, say $p_c \geq 0.6$): a random number R is uniformly sampled in (0,1] and the crossover is performed only if $R < p_c$. Vice versa, if $R \geq p_c$, the two children are copies of the parents.

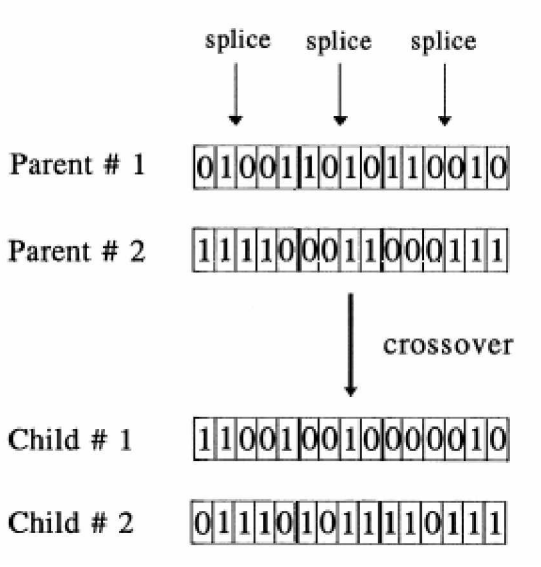

Figure 7. Crossover in a population with chromosomes constituted by three genes

The third step of the breeding procedure, performed after each generation of a pair of children, concerns the replacement in the new population of two among the four involved individuals. The simplest recipe, again inspired by natural selection, just consists in the children replacing the parents: children live, parents die. In this case, each individual breeds only once.

The fourth and last step of the breeding procedure eventually gives rise to the final $(n+1)$-th population by applying the mutation procedure to the (up to this time temporary) population obtained in the course of the preceding steps. The procedure concerns the mutation of some bits in the population, i.e. the change of some bits from their actual values to the opposite one ($0 \rightarrow 1$) and vice versa. The mutation is performed on the basis of an assigned mutation probability for a single bit (generally quite small, say 10^{-3}). The product of this probability by the total number of bits in the population gives the mean number μ of mutations. If $\mu < 1$ a single bit is mutated with probability μ. Those bits to be actually mutated are then located by randomly sampling their positions within the entire bit population.

The sequence of successive population generations is usually stopped according to one of the following criteria:

1. when the mean fitness of the individuals in the population increases above an assigned convergence value;
2. when the median fitness of the individuals in the population increases above an assigned convergence value;
3. when the fitness of the best individual in the population increases above an assigned

convergence value. This criterion guarantees that at least one individual is good enough;

4. when the fitness of the weakest individual in the population drops below an assigned convergence value. This criterion guarantees that the whole population is good enough;
5. when the assigned number of population generations is reached.

More sophisticated techniques of reproduction, crossover and replacement can be employed for a more effective search.

Furthermore, in general, the initial population sampled contains a majority of second-rate individuals together with few chromosomes, which, by chance, have moderately good fitnesses. Then, the selection rules are such that, in a few generations, almost all the moderately good chromosomes, which are actually mediocre individuals, are selected as parents and generate children of similar fitnesses; thus, almost all the second-rate individuals disappear, and most of the population gathers in a small region of the search space around one of the mediocre individuals. In this case, the genetic diversity is drastically reduced and the algorithm may achieve a premature convergence of the population fitness to a local maximum. In the course of the successive generations, the crossover procedure generates mediocre individuals, which are substituted in place of other mediocre individuals, so that the genetic selection may be seen as a random walk among mediocres. To obviate to this unpleasant premature convergence to mediocrity, a pre-treatment of the fitness function is often welcome. This is typically done by means of an affined transform of the fitnesses. Instead of applying the selection rules to the fitness $f(x)$ one works with its affined transform $f'(x)$, viz.,

$$f'(x) = a\, f(x) + b \tag{19}$$

where *a* and *b* are chosen so as to favour, at the beginning, the less fit individuals, thus maintaining genetic diversity and avoiding premature convergence [18].

5. Conclusion

These lecture notes have briefly sketched some of the concepts underlying the modern computational paradigms of neural networks, fuzzy logic systems and genetic algorithms, which are becoming of significant interest for application to condition monitoring and fault diagnostics for maintenance. Due to the limitation in the number of pages, only an intuitive and non-exhaustive treatment has been provided. Whereas some examples of practical application will be illustrated during the lecture, the interested reader is invited to refer to the specialized literature for further, in-depth details on the different techniques.

References

[1] Aven, T. (1996). Condition based replacement policies – a counting process approach. *Reliability Engineering and System Safety*, Vol. 51, 275-282.

[2] Barata, J., Guedes Soares, C., Marseguerra, M. & Zio, E. (2001). Monte Carlo simulation of deteriorating systems. *Proceedings ESREL 2001*, 879-886.

[3] Barlow, R. E. & Proschan, F. (1965). *Mathematical Theory of Reliability*. John Wiley, New York.

[4] Bérenguer, C., Grall, A. & Castanier, B. (2000). *Simulation and evaluation of condition-based maintenance policies for multi-component continuous-state deteriorating systems.* Foresight and Precaution. Cottam, Harvey, Pape & Tait (eds), Rotterdam, Balkema, 275-282.

[5] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford University Press.

[6] Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function, Mathematics of Control, Signals and Systems, Vol.2, 303-314.

[7] Dybowski, R. & Roberts, S. J. (2000). Confidence and Prediction Intervals for Feed-Forward Neural Networks, in Clinical Applications of Artificial Neural Networks, Eds. R. Dybowski and V. Gant, Cambridge University Press.

[8] Fonseca, C. M. & Fleming, P. J. (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimisation. *Evolutionary Computation*, 3(1):1-16, Spring.

[9] Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimisation, and Machine Learning.* Addison-Wesley Publishing Company.

[10] Grall, A., Bérenguer, C. & Chu, C. (1998). Optimal dynamic inspection/replacement planning in condition - based maintenance. Safety and Reliability. S. Lydersen, G. Hansen & H. Sandtorv (eds), Trondheim, Balkema, 381-388.

[11] Holland, J. H. (1975). *Adaptation in natural and artificial system.* Ann Arbor, MI: University of Michigan Press.

[12] Hontelez, J. A. M., Burger, H. H. & Wijnmalen, D. J. D. (1996). Optimum condition-based maintenance policies for deteriorating systems with partial information. *Reliability Engineering and System Safety*, Vol.51, 267-274.

[13] Klir, G. J. & Bo, Yuan (1995). *Fuzzy Sets and fuzzy logic: Theory and Application.* Prentice Hall.

[14] Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model

Selection, in C. S. Mellish Ed. *Proceedings of the 14th International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers.

[15] Kolmogorov, A. N. (1957). On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition. *Doklady Akademii Nauk SSR*, Vol. 114, 953-956.

[16] Kopnov, V. A. (1999). Optimal degradation process control by two-level policies. *Reliability Engineering and System Safety*. Vol. 66, 1-11.

[17] Lam, C. & Yeh, R. (1994). Optimal maintenance policies for deteriorating systems under various maintenance strategies. *IEEE Transactions on Reliability*. Vol. 43, 423-430.

[18] Marseguerra, M. & Zio, E. (2001). Genetic Algorithms: Theory and Applications in the Safety Domain, In "The Abdus Salam International Centre for Theoretical Physics: Nuclear Reaction Data and Nuclear Reactors", N. Paver, M. Herman and A. Gandini Eds., World Scientific Publisher, 655-695.

[19] McCall, J. J. (1965). Maintenance policies for stochastically failing equipment: a survey, Management Sci., 11, 493-524.

[20] Muller, B. & Reinhardt, J. (1991). *Neural Networks - An introduction*. Springer-Verlag, New York.

[21] Rumelhart, D. E. & McClelland, J. L. (1986). *Parallel distributed processin.*. Vol. 1, MIT Press, Cambridge, MA.

[22] Samanta, P. K., Vesely, W. E., Hsu, F. & Subudly, M. (1991). *Degradation Modelling With Application to Ageing and Maintenance Effectiveness Evaluations*. NUREG/CR-5612, U.S. Nuclear Regulatory Commission.

[23] Soares, G. C. & Garbatov, Y. (1996). Fatigue reliability of the ship hull girder accounting for inspection and repair. *Reliability Engineering and System Safety*. Vol. 51, No. 2, 341-351.

[24] Soares, G. C. & Garbatov, Y. (1999). Reliability of maintained, corrosion protected plates subjected to non-linear corrosion and compressive loads. *Marine Structures*. Vol. 12, No. 6, 425-446.

[25] Takagi, H. (1997). *Introduction to Fuzzy Systems, Neural Networks, and Genetic Algorithms, in Intelligent Hybrid Systems*. Da Ruan Ed., Kluwer Academic Publishers.

[26] Wang, W., Christer, A. H. & Jia, X. (1998). Determining the optimal condition monitoring and PM intervals on the basis of vibration analysis - A case study. *Safety and Reliability. S.* Lydersen, G. Hansen & H. Sandtorv (eds), Trondheim, Balkema, 241-246.

[27] Yeh, R. H. (1997). State-age-dependent maintenance policies for deteriorating systems with Erlang sojourn time distributions. *Reliability Engineering and System Safety*. Vol. 58, 55-60.

[28] Zadeh, L. A. (1965). Fuzzy Sets, Information and Control, Vol. 8.