Porto, Portugal, July 28-31, 2008

COMPUTATIONAL INTELLIGENCE TECHNIQUES IN ENGINEERING

VIRIATO M. MARQUES

Institute of Engineering of Coimbra
Computer Science Department, Coimbra, Portugal

viriato@isec.pt

CECÍLIA REIS

Institute of Engineering of Porto
Electrical Engineering Department, Porto, Portugal

cmr@isep.ipp.pt

LUIS ROSEIRO

Institute of Engineering of Coimbra
Mechanical Engineering Department, Coimbra, Portugal

lroseiro@isec.pt

J. TENREIRO MACHADO

Institute of Engineering of Porto
Electrical Engineering Department, Porto, Portugal

jtm@isep.ipp.pt

Computational Intelligence (CI), as defined by the IEEE Computational Intelligence Society, includes four main areas: Evolutionary Computation (genetic algorithms and genetic programming), Swarm Intelligence, Fuzzy Systems and Neural Networks.

This article shows how CI techniques overpass the strict limits of Artificial Intelligence (AI) field and can help solving real problems from distinct engineering areas: Mechanical, Computer Science and Electrical Engineering.

An introduction to each of the CI main areas is made and three systems are briefly described. The results are, in each case, very promising.

1. Introduction

Webster's New Collegiate Dictionary defines intelligence as " the ability to learn or understand or to deal with new or trying situations, the skilled use of reason and the ability to apply knowledge to manipulate one's environment or to think abstractly as measured by objective criteria (or tests)"[1].

Artificial Intelligence (AI) is the area of computer science focusing on creating machines that can engage on behaviors that humans consider intelligent. The ability to create intelligent machines has intrigued humans since ancient times and today, with the advent of the computer and 50 years of research into AI programming techniques, the dream of smart machines is becoming a reality.

Researchers are creating systems which can mimic human thought, understand speech, beat the best human chess player, and countless other feats never before possible.

Artificial Intelligence has come a long way from its early roots, driven by dedicated researchers. The beginnings of AI reach back before electronics, to philosophers and mathematicians such as George Boole and other theorists on principles that were used as the foundation of AI logic. AI really began to intrigue researchers with the invention of the computer in 1943. The technology was finally available, or so it seemed, to simulate intelligent behavior.

Alan Mathison Turing was one of the great pioneers of the computer field. He inspired the now common terms of "The Turing Machine" and "Turing's Test." As a mathematician he applied the concept of the algorithm to digital computers. His research into the relationships between machines and nature created the field of artificial intelligence.

Computational Intelligence is a successor of Artificial Intelligence. Eberhart defines computational intelligence as "a methodology involving computing (whether with a computer or wetware) that exhibits an ability to learn and/or to deal with new situations, such that the system is perceived to possess one or more attributes of reason such as generalization, discovery, association and abstraction".

Computational Intelligence (CI) as defined by the IEEE Computational Intelligence Society includes four main areas: Evolutionary Computation (genetic algorithms and genetic programming), Swarm Intelligence, Fuzzy Systems and Neural Networks. Furthermore CI is closely related to Fractals and Chaos Theory.

This article presents a brief introduction to CI main areas and describes their application to three engineering problems: 1) Neural Networks for identification of material constants 2) Fuzzy Logic for equipment fault diagnosis 3) Evolutionary Computation for digital circuit synthesis.

2. Neural Networks

It has been recognized since early that neural networks offer a number of potential benefits for application in the field of engineering, particularly for pattern recognition problems. Some appealing features of neural networks are their ability for learning through examples because, they do not require any *a priori* knowledge and they can approximate arbitrary well any non-linear continuous function [1].

Among the several architectures used in practice, the feedforward type neural networks, shown in Fig. 1, have been considered more suitable for the purposes of the signature analysis, the problem under investigation in this work

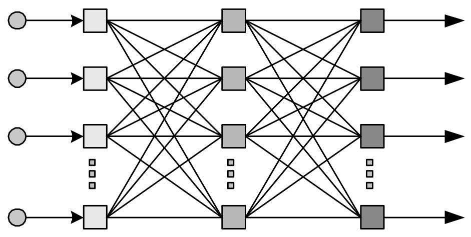


Figure 1. Feedforward neural network

A feedforward neural network consists on several layers; each one with some processing elements, called neurons, linked each other by weights. The weights determine the nature and the strength of the connection between the neurons. The number of nodes considered in the input and output layers depend on the specifications of the problem. The appropriate selection of hidden layers and its neurons is problem dependent and the optimum layout can be obtained only after extensive computational experimentation in the application domain. In fact, a too small number of the hidden units may not be sufficient to develop the required internal representation of the problem and, therefore, the neural network may not be able to perform the necessary recognition task. One the other hand, if the number of the hidden units is too large, then the network can learn to give the correct classification for all the data in the training set but with a low performance.

The choice of the activation function depends on the application to be used. Several different functions can be applied. One of the most common used is hyperbolic tangent function, represented by:

$$f(x) = -1 + \frac{2}{1+e^{-2x}} \tag{1}$$

The application of an artificial neural network consists of two stages, namely training and testing. During the training stage an input-to-output mapping, using a set of available sample data is present to the network. The learning stage stops when the maximum number of training epochs is reached. In this work a variation of the common training method, called cross validation is used. That is, a group of data, chosen in a random way among the training values, is used in each training epoch to evaluate an error function. When this error function goes different from the error with the training samples, the training process is early stopped and the node weights and bias are frozen at this point.

During the testing stage, data that has not been presented to the network in the learning stage are provided as input, and the corresponding output is calculated using the fixed node weights and bias. The evaluation of the network response to the test samples gives a measure of the correct definition and training that was performed.

In this application, feed forward type neural networks with four layers was considered. The training of the neural networks was performed with a second order type algorithm, the Levenberg Marquardt [2]. As it was pointed out by these authors, this algorithm is more efficient than other techniques when the network contains no more than a few hundred of parameters, the present situation. Cross validation is also used in the training process.

2.1. *Problem Definition*

Consider a laminated plate, made with 12 glass/epoxy equal thickness layers and with stacking sequence $[0/+45/-45/90/-45/+45]_s$. The plate have 200′ 200′ 5 *mm* and is considered simply supported on all sides. A finite element discretization with 36 equal elements is considered. Six piezoelectric sensors are placed on the plate surfaces. A schematic representation of the plate is depicted in fig. 2.

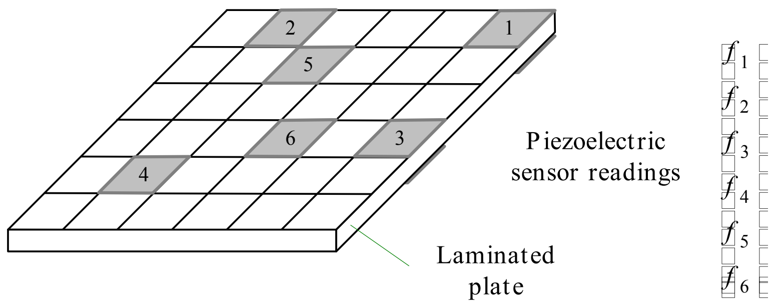


Figure 2. Laminated plate with piezoelectric sensors.

The plate is loaded with a force of 5 N applied in its center, and the sensor readings are obtained with a finite element numerical model, programmed in Matlab, and based on a displacement field with nine independent variables, given by the expressions:

$$\begin{cases} u_1(x_i,t) = u_1^0(x_\alpha,t) + x_3\ \phi_1(x_\alpha,t) + x_3^3\ \theta_1(x_\alpha,t) \\ u_2(x_i,t) = u_2^0(x_\alpha,t) + x_3\ \phi_2(x_\alpha,t) + x_3^3\ \theta_2(x_\alpha,t) \\ u_3(x_i,t) = u_3^0(x_\alpha,t) + x_3\ \phi_3(x_\alpha,t) + x_3^2\ \psi_3(x_\alpha,t) \end{cases} \tag{1}$$

where u_i^0 and ϕ_α are the displacements and the rotations in the middle plane of the plate and $\phi_3, \theta_\alpha, \psi_3$ are higher order terms [3]. It is considered the PZT 5H Morgan Matroc Piezoelectric, with the following elastic and piezoelectric properties:

$$\begin{array}{ll} E_1 = E_2 = 69GPa & E_3 = 106GPa \\ G_{12} = G_{13} = G_{23} = 26.3GPa & n_{12} = n_{13} = n_{23} = 0.31 \\ \bar{d}_{311} = \bar{d}_{322} = -171 \times 10^{-12} m/V & \bar{d}_{333} = -280 \times 10^{-12} m/V \\ \bar{p}_{33} = 1.505^{-08}\ F/m & \end{array} \tag{2}$$

The main objective is to determine the mechanical properties: E_1, E_2, G_{12} and n_{12} of a target plate, designated as "*experimental*" properties. The "*experimental*" properties and the search space to be considered are shown on table 1.

Table 1. Search space and "experimental" properties.

Properties	$E_1\,[GPa]$	$E_2\,[GPa]$	$G_{12}\,[GPa]$	n_{12}
"*experimental*"	38.48	9.38	3.41	0.292
Search space	[9.00, 58.00]	[4.00, 14.00]	[1.00, 6.00]	[0.100, 0.481]

2.2. *Neural Network Model*

The idea with this model is to use neural networks to interpret the relationship among the potential differences in the sensor and the mechanical properties associated to the laminate plate. Thus, we define four neural networks establishing a mapping between the potential differences and each property. The Fig. 3 shows an illustrative outline of the model

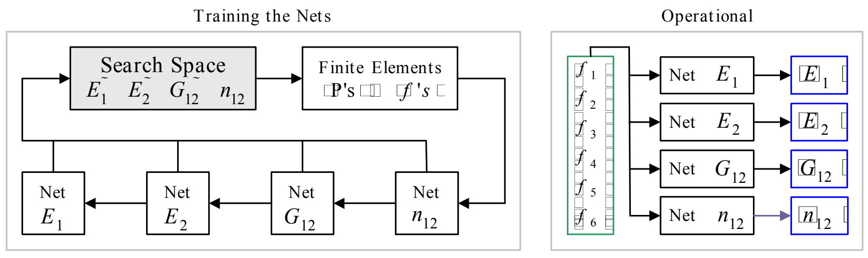


Figure 3. Neural Networks Model

In order to obtain a correct training for the neural networks and a good compromise between the number of training patterns (evaluations by the finite method) and the performance of the networks, an iterative methodology described below is used.

For the definition of the problem, the neural networks were considered with 6 neurons in the input layer, corresponding to the potential differences, and 1 neuron in the output layer, corresponding to the value of the property. The number of neurons in the internal layers (X) is adjusted iteratively during the process. Then, for each property, a network with dimension 6-X-X-1 is defined. The hyperbolic tangent function is chosen for all neurons.

An amount N, corresponding to the number of additional data for training of the nets, is defined. This amount can be seeing as a step for the methodology proposed. The maximum number of new patterns to be added to the neural networks training data, as well as the maximum number of training epochs, is also defined. A constant J, to be used as a stop criterion for this methodology, is also preset.

Inside the search space 2N combinations of properties are randomly generated. For each combination of properties it is obtained the corresponding answer of the sensors, being obtained 2N combinations of data (potential differences - properties).

Half of this data (N) is used in the training of the nets and the remaining half (N) is fixed, to be used as test data in each net. A percentage of 10% of the training data is randomly chosen for the crossed validation of each net. Then the training process of the 4 neural networks begins with these first N patterns. After this training, the outputs for the N fixed test data are request for each network and the mean relative errors between the output of each net and the fixed targets are evaluated and compared with the constant J. If the mean error value for the network is smaller than J, the training process for the corresponding network stops and the net is fixed. Otherwise, new data (N) must be added to the network.

The new data to be added are generated randomly, but guarantying that is different from the previous one. These new data is added to the other and 10% of these values are chosen for crossed validation of the nets. The dimension (X) for the number of hidden neurons is modified according to a specific criterion and a retraining step of the nets begins. After this new training, the nets are tested again with the fixed test data and an evaluation of the mean error is computed and compared with J.

This process is repeated until the mean error for all the nets is less than J. If this approach is not attained, the process stops when the maximum number for new patterns is reached.

It is important to notice that the training of the 4 nets is performed in parallel, that is, whenever new data is generated for training the nets, these are used simultaneously for all the properties where the iterative methodology still elapses. In this work, a maximum number of 90 new added data and a fixed $J = 0.3$ is defined. The maximum number of epochs is fixed as 100.

After the training of the networks, considering the values of the "experimental" properties for the plate, the corresponding potential differences in the sensors are obtained with the finite element model. Then, the output of each neural network for these "experimental" potential differences gives the properties of the plate.

2.3. *Results*

The programming of the neural networks was made in Matlab, being used a computer Pentium IV @ 2.4 MHz, 1024 Mb of RAM.

The obtained results are compared with those proposed by Liu *et al.* [4]. These authors used a SGI Origin Computer having needed 33 hours to obtain the suitable results, corresponding to 3001 evaluations of the fitness function defined for the adopted genetic algorithm. The training of the networks was stopped at fourth and sixth iteration of the methodology, respectively for the properties E_1 and E_2. For the others the 90 new patterns for training show to be necessary.

The total time of computation and the number of finite elements calculations is shown in table 2.

Table 2. Time and Number of Finite Elements Calculations.

Model	Time [s]	Number of Finite Elements Calculations
Neural Networks Model	1458	110
Liu *et al.* (2002)	120040	3001

The obtained values for the properties and the mean relative error for each property, calculated in relation to the correspondent “experimental” one, are shown in table 3.

Table 3. Results Obtained.

Model		E_1	E_2	G_{12}	n_{12}
		(38.48)	(9.38)	(3.41)	(0.292)
Neural Networks Model	Value	38.486	9.389	3.409	0.291
	Error	*0.017*	*0.101*	*0.021*	*0.362*
Liu *et al.* (2002)	Value	38.340	9.240	3.480	0.293
	Error	*0.364*	*1.493*	*2.053*	*0.342*

2.4. *Conclusions*

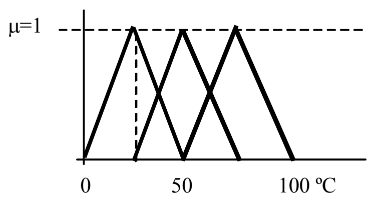
Identification of the material constants of a glass-epoxy plate has been carried out with satisfactory results. The neural network iterative model reveals to be very accurate and fast. The numerical experiments show that the iterative approach leads to small identification errors.

The obtained results are encouraging and demonstrate the effectiveness of the proposed technique to the characterization of material constants of composite structures.

3. Fuzzy Systems

A Fuzzy Set [5] is a set to which each element belongs in a degree between 0 and 1, as given by the value of its *membership function*, $\mu(x)$. This concept allows the translation of linguistic terms like *cold*, *hot*, *sometimes* or *almost always*, for instance, as shown in Fig. 4, giving rise to *Computing with Words* [6]. Fuzzy Logic, derived from fuzzy sets, is nowadays widely used in very distinct fields, from control systems to expert systems. The Mamdani and Sugeno inferences are, perhaps, one of its more divulgated results.

Possibility Theory [5, 6] is based on Fuzzy Set Theory. The distinction between probability and possibility may seem not evident: basically probability is related to occurrence frequencies while possibility expresses a (subjective) experience or judgement about something such as how much someone is *older* or *young* or how much a given quantity is *high* or *low,* for instance.



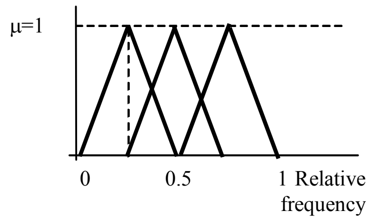


Figure 4. Examples of linguistic terms represented by fuzzy sets: "cold", "warm" and "hot" water. *Frequency Qualifiers* "rarely", "sometimes", "almost always".

Mathematically, possibility is represented by *possibility distribution functions* (see eq.3). A possibility distribution function $\pi_x(u)$, that characterizes a possibility distribution π_X associated with a linguistic term, is numerically equal to the value of the membership function $\mu_{\tilde{F}}(u)$ that defines the linguistic term.

$$\pi_X \hat{=} \mu_{\tilde{F}}(u) \tag{3}$$

(That is why in expression 3 the symbol $\hat{=}$ stands for *is defined as*.)

A possibility distribution π_X is induced whenever the Relational Assignment Equation (RAE) [6] is used to associate a linguistic term $\tilde{F}$ with the value of the attribute A of an object X (see eq.4). In this context $\tilde{R}$ acts as a fuzzy restriction, an elastic limit of the values that A may assume. An example of using expression

4 could be $\tilde{R}(temperature(oven)) = \tilde{H}$ where $\tilde{H}$ stands for the linguistic term *High*.

$$\tilde{R}(A(X)) = \tilde{F} \tag{4}$$

Given a fuzzy set $\tilde{A}$ and a possibility distribution function π_X, the *possibility measure* $\pi(\tilde{A})$ is given by [6, 7]:

$$\text{poss}\{X\ is\ \tilde{A}\} \hat{=} \sup_{u \in U} \min \{\mu_{\tilde{A}}(u), \pi_X(u)\} \tag{5}$$

The importance of eq. 5 resides on the fact that given a definition such as, $\tilde{A}$=*high_temperature*, in the presence of another fact to which corresponds another possibility distribution function, $\pi_X \hat{=} \mu_{\tilde{F}}(u)$, it is possible to calculate de possibility of X being $\tilde{A}$. For example, the possibility of *hot water* being *water at high temperature* can be computed.

Necessity measure, *N*, is related with possibility measure. The most important relation is, perhaps, the following one [6]:

$$\pi(\tilde{A}) = 1 - N(\not\subset \tilde{A}) \tag{6}$$

Necessity is a more demanding and restrictive measure than possibility in the sense that it measures how much necessarily a fact is true. The case *N=1* means that a fact is necessarily true.

Equations 5 and 6 are used in the following forms (see eq. 7, 8 and 9) in Fuzzy CLIPS [8]:

$$P(F_\beta|F_\alpha) = max(min(\mu_{F\beta}(u),\ \mu_{F\alpha}(u)) \tag{7}$$

$$N(F_\beta|F_\alpha) = 1\text{-}P(F_\beta|F_{\not\subset\alpha}) \tag{8}$$

$$\mu_{\not\subset F\alpha} = 1\text{-}\ \mu_{F\alpha} \tag{9}$$

where F_α e F_β are fuzzy sets. The format of these equations, by using the symbol "|", shows clearly that possibility and necessity of fact F_β are computed according to a previously known fact F_α.

3.1. *Problem Definition*

The principles exposed in the previous section were used in SADEX, a Case-Based-Reasoning (CBR) Fuzzy System for equipment fault diagnosis. SADEX makes use of a conceptual model of attribute values, relevance factors based on document retrieval techniques, combines information value and cost, and can perform some case adaptation tasks based on taxonomic similarities. Here we will focus on the fuzzy attribute model.

The operation cycle of a CBR system is well described by the Aamodt and Plaza diagram [9]. Basically it is composed of four phases named Retrieve, Revise, Reuse and Retain. If one relates these terms with the name Case Based Reasoning, they become almost self-explanatory: "Retrieve" involves the search and selection of past cases, more or less similar to the present case. "Reuse" may imply some kind of adaptation, so that a past solution may be applied to the present case. "Revise" makes the presentation of the (reused) solution and deals with its correctness or failure. "Retain" has to do with the recording of present cases classified as relevant for the resolution of future ones. This means learning from success or failure.

However, CBR on its own is not enough, as technical staff know-how makes effective use of subjective experiences depending on visual inspection, noise, smell and approximate measurement of some attribute values. The translation of this kind of information is possible by means of fuzzy sets and fuzzy numbers or intervals. The global similarity between cases uses fuzzy arithmetic operations and possibility theory.

3.2. *Fuzzy Model*

Equipment fault description observations must express abnormality as, if they don't, they're irrelevant. Some examples follow: *It doesn't work; Temperature is 10ºC; Temperature is low; Burned smell is evident*.

But these examples also show that abnormality can be expressed in different ways. One of the important issues in the CBR paradigm is the global similarity computation between the query case and the past cases that takes place in the *Retrieve* phase. There are many ways of evaluating this similarity but most make use of attribute values. However, in order to allow cases to be compared, a single and normalized format must be devised. Some base definitions used in this normalization process follow:

1. The *Absolute Domain* or *Possible Range* (*PR*) of an attribute is the set of

values considered of interest for abnormal facts representation. For an (L)ogical attribute *PR={0,1}*. For a (M)easurable attribute *PR=[pr_l, pr_r]*, where pr_l and pr_r stand for *PR* left and right limits; for (N)on (M)easurable (S)ubjective attributes, and due to their nature, PR is meaningless.

2. The *Absolute Typical Range* ($\tilde{ATR}$) of an attribute is a fuzzy set whose support is the set of values it can assume in *PR* in its "normal" state;

3. The *Differential Range* (*DR*) of an attribute (L, M or NMS) is a subinterval of [-1,1];

4. The *Differential Typical Range* ($\tilde{DTR}$) of an attribute is the fuzzy set that translates the linguistic term *Normal* in *DR*.

To illustrate the normalization process we will take an M-type attribute. Suppose we have *PR=[2,10]* and that $\tilde{ATR}$ is given by a fuzzy set in the α-cut notation such as (6,7,7,8). This fuzzy set defined in *PR* along with an auxiliary level and *DR* is shown in fig. 5. This figure also illustrates the important assumption that a correspondence has been established between $\tilde{ATR}$ and $\tilde{DTR}$: The fuzzy set $\tilde{ATR}$ corresponds to the linguistic term "normal" defined in $\tilde{DTR}$. This assumption has some consequences. The $\tilde{DTR}$ term is generated when, in the absolute domain, the attribute assumes the "normal value" $\tilde{v}$ or, $\tilde{v} = \tilde{ATR} = (a,b,c,d)_{ATR}$. In this case we will get a difference $\Delta \tilde{ATR}$ given by:

$$\Delta \tilde{ATR} = \tilde{ATR} - \tilde{ATR} = (a,b,c,d)_{ATR} - (a,b,c,d)_{ATR} = (a-d, b-c, c-b, d-a)_{ATR} \qquad (10)$$

This fuzzy interval is represented at the auxiliary intermediate level of fig. 5.

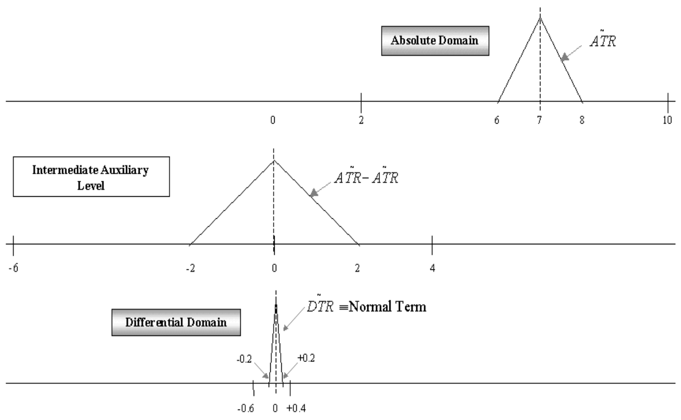

Absolute Domain
$\tilde{ATR}$
0
2
6
7
8
10
Intermediate Auxiliary Level
$\tilde{ATR}-\tilde{ATR}$
-6
-2
0
2
4
Differential Domain
$\tilde{DTR}$ =Normal Term
-0.2
+0.2
-0.6
0
+0.4

Figure 5 - The Typical Range of a M-type attribute in the Absolute and Differential Domains

As *a-d* is always symmetrical of *d-a* and *b-c* of *c-b*, we verify that $\Delta\tilde{ATR}$ has always a zero medium value and is symmetrical, independently of the shape that $\tilde{ATR}$ may assume. Besides this, the maximum differences allowed between any possible attribute value and its $\tilde{ATR}$ are $pr_l - \tilde{ATR}$ for the left (lower) limit, and $pr_r - \tilde{ATR}$ for the right (upper) limit. So, the limits for the fuzzy set represented in the auxiliary level of fig. 5, α and β, are given by:

$$\alpha = \min\left(pr_l - \tilde{ATR}\right) = pr_l - d_{ATR} \qquad (11)$$

$$\beta = \max\left(pr_r - \tilde{ATR}\right) = pr_r - a_{ATR}\,.$$

In our example we get α=-6 and β=4. We have, then, an "intermediate domain" whose width, *w*, is given by

$$\begin{aligned} w &= \left|pr_l - d_{ATR} - \left(pr_r - a_{ATR}\right)\right| = \\ &= \left(pr_r - pr_l\right) + \left(d_{ATR} - a_{ATR}\right) \end{aligned} \qquad (12)$$

So, *w* is the value by which we must divide $\Delta\tilde{ATR}$ in order to normalize it and to obtain the "normal term" in the differential domain, $\tilde{DTR}$. Defining θ as

$$\theta = (pr_r - pr_l) + (d_{ATR} - a_{ATR}) \tag{13}$$

then combining equations 10 and 13 it yields:

$$\begin{aligned}\tilde{DTR} &= (a,b,c,d)_{DTR} = \\ &= (a-d, b-c, c-b, d-a)_{ATR} / \theta\end{aligned} \tag{14}$$

This gives the fuzzy interval *(-0.2, 0, 0, 0,2)*that represents the "normal" linguistic term shown at the *differential domain* of fig. 5. This also implies that the limits of the differential domain are variable and depend on the location of $\tilde{ATR}$ in *PR*. In fact, let *DR=[dr$_l$, dr$_r$]*. Combining expressions 11 and 13, we have:

$$\begin{aligned}dr_l &= \alpha/\theta = (pr_l - d_{ATR})/\theta \\ dr_r &= \beta/\theta = (pr_r - a_{ATR})/\theta\end{aligned} \tag{15}$$

For our example we get *dr_l=-0.6* and *dr_r=0.4*, visible at the differential domain of fig. 5.

Every time a system user makes an attribute definition by specifying some of the elements above described, an weighted mean of all its past definitions with the new one is made (for each similar equipment group). The results are sets of linguistic terms each one defined by a fuzzy set that attempt to catch the concept *linguistic terms usually used to describe abnormal attribute values for a given equipment group*. We call these sets of fuzzy sets *Conceptual Models* that work as templates when a new definition is made. The adaptation of original values to a new definition is called *Adaptation*. Finally the *Projection* operation is symmetrical to *normalization* and simulates the reality view as modeled by previously defined concepts.

According to this, attribute representation and handling can be conveniently supported by the 3-Level Model (3L Model) shown in fig. 6. This model is composed of three levels (External, Operational and Conceptual) related with the attribute domain and human perception:

1. External Level / Absolute Domain / Reality - models the reality, the real world;
2. Operational Level / Differential Domain / Human Cognition (reasoning) - handles normalized attribute values and supports every computation;
3. Conceptual Level / Conceptual Domain - / Human Cognition (learning) - handles meta-models for future attribute definitions.

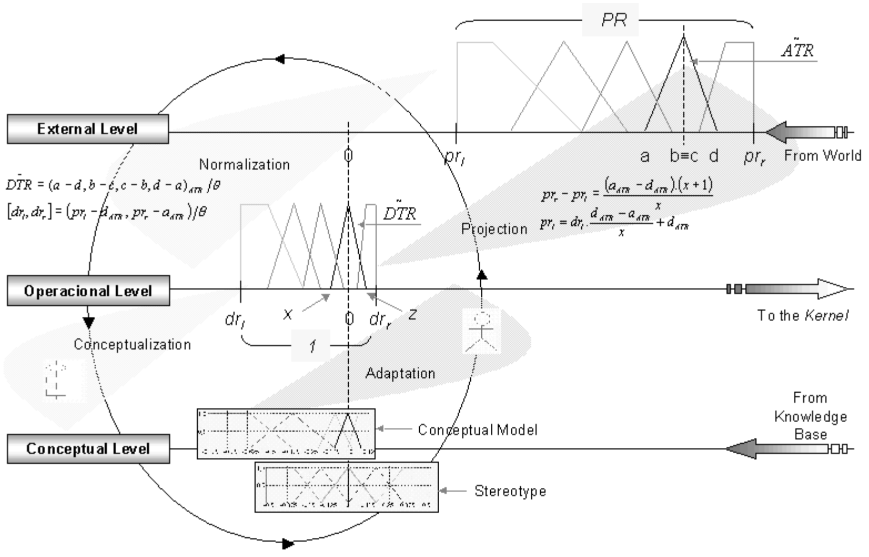


Figure 6. A 3-Level Model for attribute handling

This 3L Model allows any observation to be translated, after some manipulation, by a fuzzy interval whose x-axis values lie in the interval [-1, 1] along with an attribute ID and an equipment group ID. This basic application of RAE can be complemented with qualifiers and quantifiers that help to express occurrence frequency and refer to several equipment components. Combining all these elements a Canonical Observation Form (COF) has been defined, as follows:

[Qf] (Attribute ([Qt] ([CompID](EquipID)))) =
= [NOT], [Mod.], Value, [Unit])

- *Qf - Frequency Qualifier*
- *Qt – Quantifier*
- *CompID – Component ID*
- *EquipID – Equipment ID*
- *NOT – Logical operator*
- *Mod. - Modifier*
- *Value - Normalized attribute value*
- *Unit. - Measurement unit*
- *[..] stands for "optional"*

A set of COFs describes a case and COFs are represented at the Operational Level where they are used to compute global case similarity.

The diagnosis process computes the global similarity between a query case and relevant past cases selected from the Case Library. This computation generates one positive and a maximum of three negative contributions for every diagnosis taken into account:

The positive contribution is generated by corresponding observations that both represent deviations form normal values (error signals) of the same sign, that is, both positive or both negative. This feature implements *diagnosis selection by present symptom in the query case Q and the past case* P_i. This positive contribution can be emphasized or depreciated according to the value of an weighting factor R_{ij} that expresses the relevance of observation i to a possible diagnosis j:

$$R_{ij} = \frac{CF_{ij}.OF_i}{NSL_j} \tag{16}$$

Where CF_i expresses by how many different diagnosis the same observation spreads (Collection Frequency), OF_{ij} represents how frequently each diagnosis generates a given observation (Observation Frequency), NSL_j characterizes how many observations each diagnosis generates when compared to the average of all the known diagnosis (Normalised Syndrome Length). Equation 16 is inspired by the document retrieval technique described in [10].

The first negative contribution is also generated by the corresponding observations that represent error signals of the same sign, but in which the attribute value in Q is somewhat *normal*. For this negative contribution a factor of the type *how much a value is normal* is computed according to the *fuzzy pattern matching* technique described in [6]. This feature implements the *diagnosis exclusion by absent symptom in Q and present in* P_i.

The second negative contribution is generated by the corresponding observations that represent error signals of contrary signs. This feature implements the *diagnosis exclusion by present symptom in Q and absent in* P_i or vice-versa.

Absent descriptions in Q or in P_i are also taken into account and can lead to other negative contribution or determinate the beginning of an *ask-for-new-observation* cycle, that takes into account relevance and cost as estimated by the technical staff.

3.3. *Results*

A prototype was implemented according to the principles exposed in the

Proceedings of the 2nd Conference on Nonlinear Science and Complexity
Porto, Portugal, July 28-31, 2008

above sections and allowing Internet access. Other techniques such as transformational case adaptation was used too as, under certain circumstances, this adaptation allows the extension of a known diagnosis for equipment A to a similar equipment B, based on taxonomic knowledge [11, 12].

The system has been tested in the health-equipment maintenance field. The case library was initially loaded with occurrences contained in work orders of SUCH - Serviços de Utilização Comum dos Hospitais. The results are promising with a correct diagnosis near 100% in some cases.

3.4. *Conclusions*

Equipment fault diagnosis was carried out with satisfactory results. The fuzzy model for attribute representation and handling proved to be adequate for technical staff daily maintenance tasks and the system can be useful for personnel training too. However, a complete field test is still needed as CBR systems, by their nature, need a considerable time period for correct tuning and evaluation.

4. Evolutionary Algorithms

In recent decades Evolutionary Computation (EC) techniques have been applied to the design of electronic circuits and systems, leading to a novel area of research called Evolutionary Electronics (EE) or Evolvable Hardware (EH) [13]. EE considers the concept for automatic design of electronic systems. Instead of using human conceived models, abstractions and techniques, EE employs search algorithms to develop implementations not achievable with the traditional design schemes, such as the Karnaugh or the Quine-McCluskey Boolean methods.

This section presents an application of an Evolutionary Algorithm, a Genetic Algorithm (GA), to the design of combinational logic circuits.

GAs are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetic. The basic concept of GAs is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem [14].

First pioneered by John Holland in the 60s, GAs has been widely studied, experimented and applied in many fields in engineering worlds. Not only does

GAs provide alternative methods to solving problem, it consistently outperforms other traditional methods in most of the problems.

GAs were introduced as a computational analogy of adaptive systems. They are modeled loosely on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of operators such as mutation and recombination (crossover). A fitness function is used to evaluate individuals, and reproductive success varies with fitness.

In this specific case a GA strategy is adopted to design digital circuits, in particular combinational logic circuits. Therefore, it is necessary to define the population of individuals in the form of chromosomes (circuit encoding) that represent the digital circuits. Another important issue that must be well established is the fitness function that will be responsible for the evaluation of the circuits.

The next sub-sections present the problem definition, the circuit encoding, the genetic operators employed and the fitness function applied.

4.1. *Problem Definition and Circuit Encoding*

The circuits are specified by a truth table and the goal is to implement a functional circuit with the least possible complexity. Since the combinational logic circuits are composed by logic gates, for this study were defined four sets of logic gates, as shown in table 4, being Gset 2 the simplest one and Gset 6 the most complex gate set. Logic gate named WIRE means a logical no-operation.

Table 4 Gate sets.

Gate Set	Logic gates
Gset 6	{AND,OR,XOR,NOT,NAND,NOR,WIRE}
Gset 4	{AND,OR,XOR,NOT,WIRE}
Gset 3	{AND,OR,XOR,WIRE}
Gset 2	{AND,XOR,WIRE}

In the presented scheme the circuits are encoded [15] as a rectangular matrix **A** ($row \times column = r \times c$) of logic cells (fig. 7). Three genes represent each cell: <*input1*><*input2*><*gate type*>, where <*input1*> and <*input2*> are one of the circuit inputs, if they are in the first column, or one of the previous outputs, if they are in other columns. The gate type is one of the elements adopted in the gate set. The chromosome is formed with as many triplets as the matrix size demands (*e.g.*, triplets = $3 \times r \times c$).

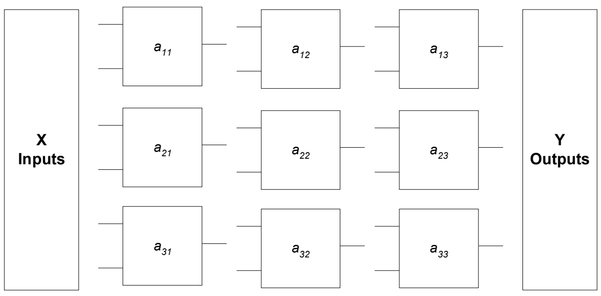


Figure 7. A 3 × 3 matrix **A** representing a circuit with input **X** and output **Y**.

4.2. *The Genetic Operators*

The initial population of circuits (strings) is generated at random. The search is then carried out among this population. The three different operators used are reproduction, crossover and mutation, as described in the sequel.

In what concern the reproduction operator, the successive generations of new strings are reproduced on the basis of their fitness function. In this case, it is used a tournament selection to select the strings from the old population, up to the new population.

For the crossover operator, the strings in the new population are grouped together into pairs at random. Single point crossover is then performed among pairs. The crossover point is only allowed between cells to maintain the chromosome integrity.

The mutation operator changes the characteristics of a given cell in the matrix. Therefore, it modifies the gate type and the two inputs, meaning that a completely new cell can appear in the chromosome. Moreover, it is applied an elitist algorithm and, consequently, the best solutions are always kept for the next generation.

To run the GA we have to define the number of individuals to create the initial population *P*. This population is always the same size across the generations, until the solution is reached.

The crossover rate *CR* represents the percentage of the population *P* that reproduces in each generation. Likewise, the mutation rate *MR* is the percentage of the population *P* that can mutate in each generation.

4.3. *The Fitness Function*

The calculation of the fitness function F in (17) has two parts, f_1 and f_2, where f_1 measures the functionality and f_2 measures the simplicity. In a first phase, we compare the output **Y** produced by the GA-generated circuit with the required values $\mathbf{Y_R}$, according with the truth table, on a bit-per-bit basis. By other words, f_1 is incremented by one for each correct bit of the output until f_1 reaches the maximum value f_{10}, that occurs when we have a functional circuit. Once the circuit is functional, in a second phase, the algorithm tries to generate circuits with the least number of gates. This means that the resulting circuit must have as much genes *gate type* ≡ *wire* as possible. Therefore, the index f_2, that measures the simplicity (the number of null operations), is increased by *one* (zero) for each wire (gate) of the generated circuit, yielding:

- First phase, circuit functionality:

$$f_{10} = 2^{ni} \times no \tag{17a}$$

$$f_1 = f_1 + 1, \text{ if } \{\text{bit } i \text{ of } \mathbf{Y}\} = \{\text{bit } i \text{ of } \mathbf{Y_R}\}\ ,\ i = 1, \ldots, f_{10} \tag{17b}$$

- Second phase, circuit simplicity:

$$f_2 = f_2 + 1 \text{ if } \textit{gate type} = \textit{wire} \tag{17c}$$

$$F = \begin{cases} f_1, & F < f_{10} \\ f_1 + f_2, & F \geq f_{10} \end{cases} \tag{17d}$$

where *ni* and *no* represent the number of inputs and outputs of the circuit.

4.4. *Results*

This section shows the implementation of four different combinational logic circuits, namely, a 2-to-1 multiplexer, a one-bit full adder, a four-bit parity checker and a two-bit multiplier, using the GA algorithm.

Due to the stochastic nature of the GAs in order to evaluate its performance, for each gate set we perform 20 simulations. Table 5 shows the average of the number of generations μ(N) to reach the solution and the average of the fitness function μ(F) after performing 20 experiments for each gate set.

Table 5 Results for the 2-to-1 multiplexer, the one-bit full adder, the four-bit parity checker and the two-bit multiplier circuits, using the GA algorithm.

	Circuit							
Gate set	**2-1 Multiplexer**		**1-bit Full Adder**		**4-bit Parity Checker**		**2-bit Multiplier**	
	μ(N)	***μ(F)***	***μ(N)***	***μ(F)***	***μ(N)***	***μ(F)***	***μ(N)***	***μ(F)***
Gset 6	27,15	10,25	72,45	18,15	32,55	21,70	1699,00	69,15
Gset 4	19,75	10,35	53,65	18,35	20,40	21,95	1183,05	69,50
Gset 3	13,55	10,5	32,40	18,45	13,754	22,65	432,40	70,25
Gset 2	12,05	11,5	34,86	18,57	7,95	23,95	362,35	70,45

It is possible to see the superiority of the gate sets 2 and 3 in terms of μ(N) and μ(F).

Figure 8 illustrates the average of the fitness function μ(F) *versus* the average of the number of generations μ(N) to reach the solution, for all the gate sets (*i. e. Gsets* 2, 3, 4 e 6) and for all the circuits.

Comparing the four studied cases, based on the average of the number of generations μ(N) to reach the solution and the average of the fitness function μ(F) it is possible to conclude that, independent of the circuit complexity, the best results appeared with the reduced gate sets.

In brief, this application uses a GA for designing combinational logic circuits given a set of logic gates. The final circuit is optimized in terms of complexity (with the minimum number of gates).

For all the case studies the GA has proved to be efficient, even when the number of outputs in the truth table increases. It is also visible that the performance of the GA increases as the complexity of the gate set decreases.

Experiments show that we have better results with *Gset* 2, that is, the simplest set that we have adopted in this study.

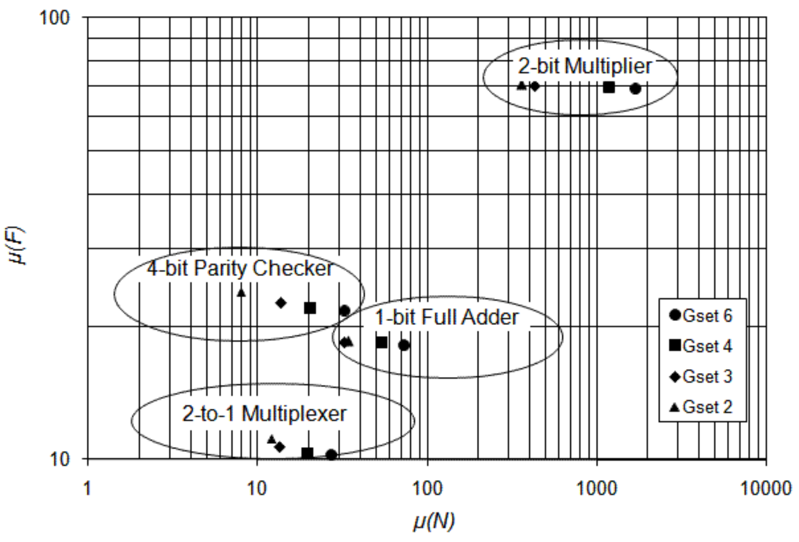


Figure 8: Average of the fitness function μ(F) *versus* the average of the number of generations μ(N) to reach the solution, using the GA algorithm.

5. Conclusion

As stated before, we show how CI techniques overpass the strict limits of AI field and can help solving real problems from distinct engineering areas: Mechanical, Computer Science and Electrical Engineering.

For AI researchers these are good news! Still, as a relatively new disciplinary field, AI has a long way to go. "Proud, not smug", according to Menzies [16]; or, "take pride in how far you have come; have faith in how far you can go" (Anonymous).

Acknowledgments

The authors would like to acknowledge the GECAD and the CEMUC Research Units.

References

1. Hornik, K., H. Stinchcombe and White, H., (1989), Multilayer Feedforward Networks are Universal Approximators, Neural Networks, Vol. 2, 183-192.
2. Hagan, T., Demuth, H.B. and M. Beale, Neural Network Design, (1996), PWS Publishing Company, USA.
3. Allik, H. and Hughes, T.J., (1970), Finite Element Method for Piezoelectric Vibration, International Journal for Numerical Methods in Engineering, Vol. 2, 151-157.
4. Liu, G.R., Ma, W.B. and. Han, X., (2002), An inverse procedure for determination of material constants of composite laminates using elastic waves, Computational. Methods in Applied Mechanics and Engineering, Vol. 191, 3543-3554.
5. Bellman,R.E., Zadeh,L.A., 1977, Local and Fuzzy Logics, Selected Papers by Lofti A. Zadeh, 1965-1996, Advances in Fuzzy Systems - Applications and Theory, Vol.6, World Scientific, 1996.
6. Zimmerman ,H.J., Fuzzy Set Theory and it's Applications, Kluwer Academic Publishers, 1996.
7. Zadeh,L.A., 1978, Fuzzy Sets as a Basis for a Theory of Possibility, Fuzzy Sets and Systems 1.
8. NRCC-National Research Council of Canada - Institute for Information Technology, (1983), Fuzzy CLIPS version v6.04 A.
9. Aamodt,A., Plaza,E., (1994), Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches, AI-Com - Artificial Intelligence Communications, Vol7, 1.
10. Robertson,S., Spark-Jones,K., (1997), Simple, Proven Approaches to Text Retrieval, Department of Information Science - City University - Cambridge - United Kingdom.
11. Bergmann,R., (1998), On the Use of Taxonomies for Representing Case Features and Local Similarity Measures, Proceedings of the 6th German Workshop on Case-Based Reasoning GWCBR'98.
12. Bergmann,R., Stahl,A., (1998), Similarity Measures for Object-Oriented Case Representations, Proceedings of the European Workshop on Case-Based Reasoning, EWCBR'98.
13. Zebulum, R. S., Pacheco, M. A. and Vellasco, M. M., Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms, CRC Press, 2001.
14. Goldberg D E (1989) Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley.

15. Cecília Reis, J. A. Tenreiro Machado, and J. Boaventura Cunha (2004) Evolutionary Design of Combinational Logic Circuits, JACIII, Fuji Tec. Press, Vol. 8, No. 5, pp. 507-513, Sept.
16. Menzies, T., (2003), IEEE Intelligent Systems (http://menzies.us/ pdf/03aipride.pdf).