OCCASION

This publication has been made available to the public on the occasion of the 50th anniversary of the United Nations Industrial Development Organisation.

DISCLAIMER

This document has been produced without formal United Nations editing. The designations employed and the presentation of the material in this document do not imply the expression of any opinion whatsoever on the part of the Secretariat of the United Nations Industrial Development Organization (UNIDO) concerning the legal status of any country, territory, city or area or of its authorities, or concerning the delimitation of its frontiers or boundaries, or its economic system or degree of development. Designations such as "developed", "industrialized" and "developing" are intended for statistical convenience and do not necessarily express a judgment about the stage reached by a particular country or area in the development process. Mention of firm names or commercial products does not constitute an endorsement by UNIDO.

FAIR USE POLICY

Any part of this publication may be quoted and referenced for educational and research purposes without additional permission from UNIDO. However, those who make use of quoting and referencing this publication are requested to follow the Fair Use Policy of giving due credit to UNIDO.

CONTACT

Please contact publications@unido.org for further information concerning UNIDO publications.

For more information about UNIDO, please visit us at www.unido.org

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION
Vienna International Centre, P.O. Box 300, 1400 Vienna, Austria
Tel: (+43-1) 26026-0 · www.unido.org · unido@unido.org

XD9700193

21751

48p.
tables
graphs
diagrams

INTELLIGENT TECHNOLOGIES. NEW OPPORTUNITIES FOR MODERN INDUSTRY.

Alexander R. Kornilov — Consultant
Backstop Eff. No. Phind

Moscow Institute of Electronic Technology - Technical University
Computer Science Department.

1. Introduction

During the last few decades the attitude of the industrial community to the ability of computer science to deal with uncertainty of data in a way any human being does, has changed from unlimited optimism to a more realistic and pragmatic approach. Their ambitions shifted from creation of artificial intelligence towards development of intelligent technologies. The latter are usually considered as computer methods to solve problems that require any intellectual efforts. The scope of possible applications of intelligent technologies is very wide : automatic control, decision making, learning, natural languages, scheduling, speech and pattern recognition, games etc.

Studying modern trends in intelligent technologies reveals an increasing role of "Soft Computing" as the main methodology of design and analysis of intelligent systems. The term "Soft Computing" introduced by L.Zadeh [1] means an approach based on computer algorithms driven by their own inseparable guiding principle in contrast to traditional "Hard Computing" driven by some deterministic algorithm (program). The main activities in "Soft Computing" are centred around the following principal areas : fuzzy logic, neural networks and genetic algorithms (GAs).

The general objective of soft computing methods is to provide adaptation to changes of the analysed object behaviour, what is inherent to many real-life problems. Here abundance, fuzziness and incompleteness of knowledge are not disadvantages to avoid but essential parts of a domain model to obtain the optimal or a sufficient solution within a shorter time period and with less efforts (expenditure) comparing with traditional methods.

The relations between microelectronics and intelligent technologies can be specified as special or mutually beneficial. There are two reasons for their partnership. On the one hand, microelectronics as industry and science, requires carrying out R&D and production methods matching its sophistication. Indeed, the following specific features of microelectronics prove it:

systems of microelectronics (either these are their products or technological equipment or technological processes or processes of R&D) are so complex that there are no precise deterministic or stochastic mathematical models available. It can occur when some factors affecting a process or their contribution are unknown;

time consumed by precise mathematical methods are intolerable;

critical importance of human factor in microelectronic manufacturing that can hardly be captured by traditional methods;

extremely high reliability of products of microelectronics makes it practically impossible to obtain data samples in a sufficient volume;

some parameters of technological process of microelectronics allow only qualitative representation;

at the beginning of a design process a great deal of expert estimation, discourses in form of tendencies, heuristics are available along with no need in high precision of decisions made at this stage.

On the other hand, the progress of microelectronics opens new opportunities for application of microelectronic hardware, stimulates the development of new methods of complex systems analysis and does provide manufacturers and designers with appropriate hardware to implement modern intelligent technologies.

2.1. Fuzzy logic

2.1.1. Basic concepts

Fuzzy Logic was initiated in 1965 by Lotfi A. Zadeh [2], professor for computer science at the University of California in Berkeley, and it is probably the most developed part of intelligent technologies. This theory is based on extension of traditional (Boolean) binary logic which is considered inadequate for capturing a domain uncertainty, ambiguities or a number of exceptions.

Fuzzy Logic is basically a multivalued logic that allows intermediate values to be defined between conventional evaluations like yes/no, true/false, black/white etc. Notions like high or pretty high can be formulated mathematically and processed with a computer. For example, is 50 degrees Centigrade a high or low temperature of integrated circuit performance environment? In the real world an answer "it depends" is common, and in fuzzy logic "some of both" might be the answer, that is 50 degrees Centigrade is partially high and partially low temperature. In this way an attempt is made to apply a more human-like way of thinking in the programming of computers. Fuzzy logic in this sense provides a designer with a powerful technology of presenting expert knowledge in a more natural and explicit way almost like in the daily life.

This is reached by a more general notion of a set comparing to the classical set theory. In classical or conventional set theory, the set S is defined by a function fs called a characteristic function of S. It maps elements of S to $\{0,1\}$, so that for $x \in S$,

$$f_s(x) = \begin{cases} 1, & if\ x \in S \\ 0, & if\ x \notin S \end{cases}$$

Therefore, for any element x of S $fs(x)=1$, if x is an element of S, and $fs(x)=0$, if x is not an element of S.

In contrast, in fuzzy set theory the set *S* is defined by a function *μs*, called a membership function of *S*. This function maps elements of *S* to *[0,1]*, so that for $x \in S$, $\mu s=1$ means that *x* totally belongs to *S*, $\mu s=0$ means that *x* does not belong to *S*, and $0<\mu s<1$ means that *x* belongs to *S* with a degree of membership of *μs*.

Therefore fuzzy logic recognises not only "pure" cases (belongs/does not belong), but also infinite gradations in between, assigning numbers (degrees of membership) to them. So, if the membership function of "High temperature" is as in Fig. 1, then 75°C might be classified as high with a degree of 0.7. Such numbers can be used to obtain exact solutions using imprecise information.

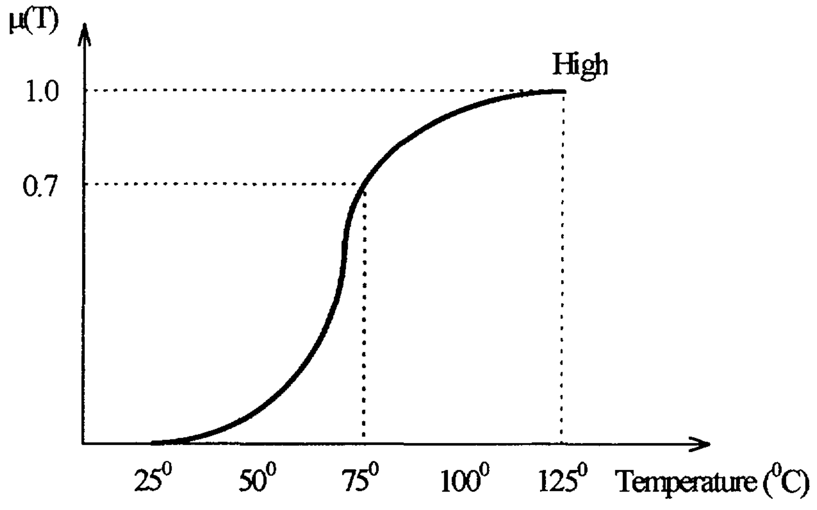


Fig.1. Fuzzy set "High temperature"

2.1.2. Fuzzy decision making (control) process.

Within fuzzy approach these expressiveness and inherent flexibility are reached by special logic inference on a set of "IF-THEN" rules which combine antecedents (inputs) and consequences (respective outputs) in a way to match a domain (or at least an expert knowledge of it). A set of such rules is called a knowledge base. A control (decision making) procedure in fuzzy systems usually consists of the following steps: fuzzification, rule evaluation, defuzzification [3].

Fuzzification is a procedure of obtaining fuzzy representation of crisp values using predefined set of membership functions. As soon as such set is defined the procedure of fuzzification compares crisp values with the membership functions to determine corresponding fuzzy values. So the first step in fuzzification procedure is to define a linguistic variables [4], i.e. variables with values form a set of artificial or natural language sentences. These sentences are considered as labels of fuzzy sets. For example, a linguistic variable "*temperature*" can have values "*small*" - *S*, "*less then medium*" - nM, "*medium*" -M, "*more then medium*" - pM, "*high*" - H.

At the next step of fuzzification for each label of fuzzy sets, corresponding membership functions are defined. Each membership function identifies the range of input values that corresponds to a label. Although a precise deterministic procedure is used to process fuzzy rules, fuzziness is introduced by configuring overlap between antecedent's membership function domain, so crisp input values can belong to more than one fuzzy set. Fuzzy outputs also have membership functions. Describing crisp inputs in fuzzy terms allows the system to gracefully respond to gradual changes in input. The shape of the membership function reflects subjective opinion of an expert about the system behaviour, or states and affects the fuzzy process a in subtle way. For example, the function's shape directly affects the time and space requirements for a microcontroller performing fuzzification and defuzzification. Membership functions can be of several different shapes (Fig. 2).

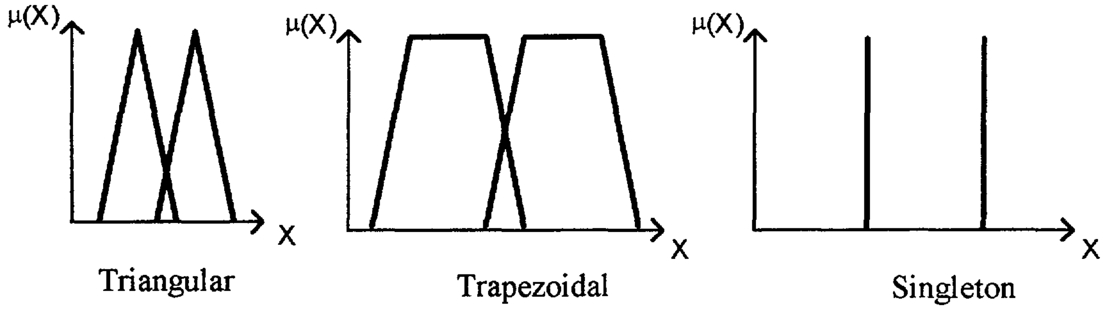


Fig.2. Membership function shapes.

Trapezoidal and triangular shapes of membership functions are most frequently used. Membership functions of arbitrary shapes may be more representative, but they require more complicated equations or large lookup tables to be represented accurately. Singletons are easily represented in computer and require more simple defuzzification algorithms. So they are frequently used to represent fuzzy outputs. An example of membership functions of fuzzy sets for the linguistic variable "*temperature*" is shown in Fig. 3.

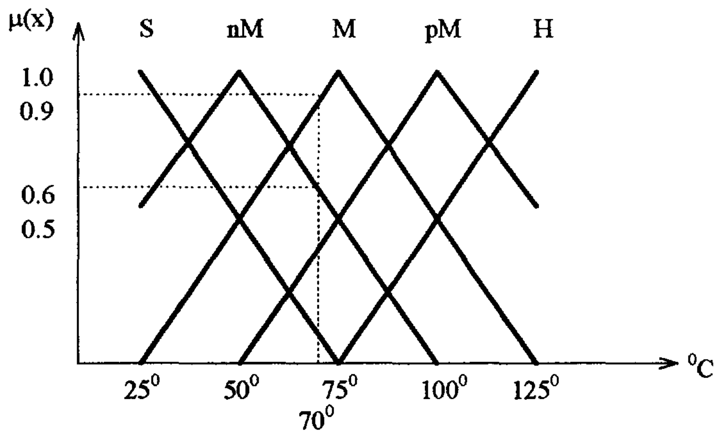


Fig.3. Membership functions for linguistic variable "*temperature*".

Depending on the shapes of membership functions, various methods are used to represent the functions in a microcontroller. A point-slope representation allows trapezoidal, triangular, and singleton functions to be represented with a minimal amount of space and time. A lookup table is a common representation for an arbitrary shaped function. It is the fastest representation in terms of fuzzification, but it requires a lot of memory. Studies have shown that reasonable performance and the significant time savings can be obtained using singleton outputs, but the resultant output actions may not represent the response as closely as desired.

Rule evaluation (fuzzy inference) is the second stage of the fuzzy logic processing where linguistic rules are used to determine what fuzzy outputs (control actions, decisions) should occur in response to a given set of input values. In processing of a set of rules, the fuzzy system (algorithm) fires rules with antecedents which membership functions indicate non-zero degree of membership corresponding to a fuzzy inputs generated by a current crisp value of a system input on the stage of fuzzification. So several rules can be fired at the same time. Fuzzy rules are usually IF-THEN statements that describe the action to be taken in response to various fuzzy inputs. Linguistic rules are confined to a predefined set of linguistic terms and a strict syntax:

IF *antecedent 1* AND *antecedent 2* ..., THEN *consequent 1* AND *consequent 2* ... ,

where AND is one of the fuzzy logic operators, and the antecedent and the consequent are in the form of:

Antecedent:	*Input variable **is** label,*
Consequent:	*Output variable **is** label.*

Rules describe the behaviour of a system and are written in terms of the membership function linguistic labels. For two-input one-output system the rules can be represented in the form of a table in Fig. 4, where an algorithm of evaluation of mean-time to failure of integrated circuit metallization is presented. Here $\tilde{I}$ and $\tilde{T}$ mean linguistic variables "*current density*" and "*temperature*", respectively.

$\tilde{I}$ \ $\tilde{T}$	S	nM	M	pM	L
S					
nM					M
M			M		
pM		pM			S
L					

Fig.4. Fuzzy algorithm for evaluation of mean-time to failure of metallization.

This fuzzy algorithm consists of 4 linguistic rules:

1) IF *current density* is pM AND *temperature* is nM,

THEN *mean-time to failure of metallization* is pM.

2) IF *current density* is pM AND *temperature* is H,

THEN *mean-time to failure of metallization* is H.

3) IF *current density* is nM AND *temperature* is H,

THEN *mean-time to failure of metallization* is M.

4) IF *current density* is M AND *temperature* is M,

THEN *mean-time to failure of metallization* is M.

The next step in the procedure of fuzzy inference is to evaluate the relevance or degree of membership of each rule antecedent. The relevance of an antecedent is determined as a degree of membership of an input crisp value to a fuzzy set of the corresponding antecedent. This is performed for each antecedent of a rule.

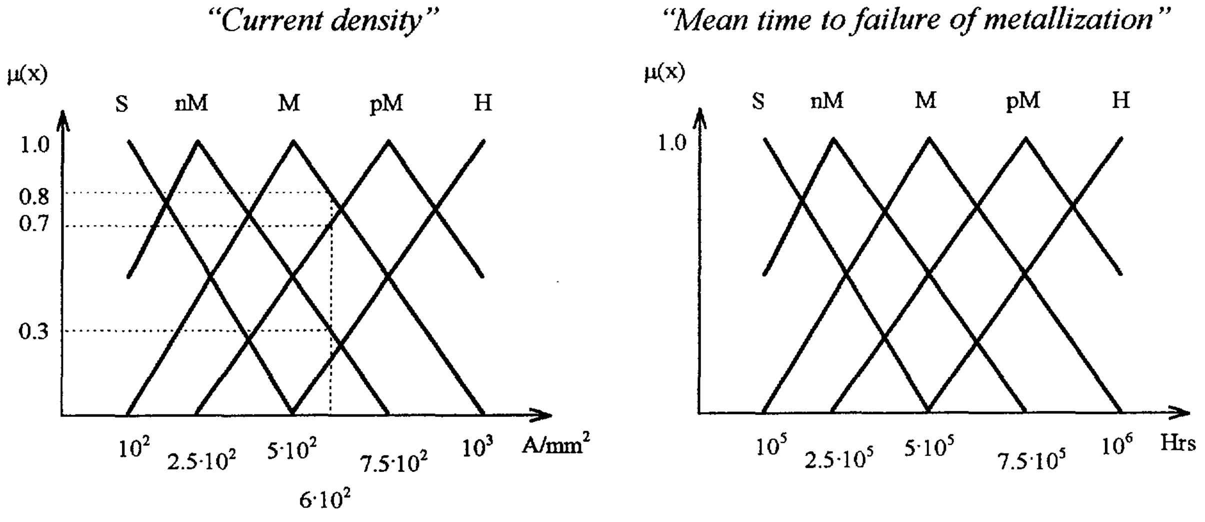


Fig.5. Fuzzy sets used in algorithm of evaluation of mean-time to failure of metallization

Let the linguistic variables for the fuzzy algorithm of evaluation of mean-time to failure of integrated circuit metallization be defined as in Fig. 4, with fuzzy sets of variables "current density" and "mean-time to failure of metallization" presented in Fig.5. Then for the linguistic variable "current density" and inputs of 6.102 A/mm2 of current density the relevance of antecedents are 0.7 for the fuzzy set "pM", 0.3 for the fuzzy set "nM" and 0.8 for the fuzzy set "M" (Fig.5). Following the same procedure for the linguistic variable "temperature" and input of 70°C, the relevance of antecedents are 0.6 for the fuzzy set "nM", 0.9 for the fuzzy set "M" and 0.0 for the fuzzy set "H" (Fig.3).

At the following step the degree of truth (rule strength) for each rule is to be determined as the smallest strength value (relevance) of the rule antecedents. The strengths of the rules of the algorithm presented above, when crisp input values are 6.102 A/mm2 a current density and 70°C for temperature, are:

1) min{0.7,0.6}=0.6

2) min{0.7,0.0}=0.0

3) min{0.3,0.0}=0.0

4) min{0.8,0.9}=0.8.

The next step is to determine the fuzzy output by comparing the rule strengths of all rules that represent the same consequent label (output action). When there are several rules with the same consequent label, the fuzzy output is determined as the largest rule strength of all such rules. This procedure produces one fuzzy output membership function label. For fuzzy algorithm presented in Fig.4, the fuzzy output for the given crisp inputs (6.105 A/cm2, 70°C) will be

mean -time to failure of metallization is pM (0.6) OR M (0.8).

So the main steps of the rule of inference described (min-max inference) are the following:

- describe the system behaviour in terms of IF-THEN rules;

- apply fuzzification procedure for current crisp input values to determine the degree of truth of each antecedent:

- determine the strength of the entire rule as the minimum of the antecedent degrees of truth;

- find the fuzzy outputs which are equal to the maximum rule strength for each consequent label.

During the last stage - **defuzzification** - a single crisp output action is obtained. There are mainly two algorithms: the mean of maxima and the centre of gravity procedures. The first one takes the smallest x1 and the largest x2 of the elements which give the maximum membership value, and calculate the mean of the two.

One of the most frequently used defuzzification method is the centre of gravity (COG) or centroid method. According to this method each output membership function is truncated above the value determined by its respective fuzzy output. Since all resulting membership functions are determined, they are combined according to OR operator as the union of corresponding fuzzy sets. The union of two fuzzy sets A and B with membership functions *μA(x)* and *μB(x)*, $x \in U$ is a fuzzy set $A \cup B$ with the membership function $\mu A \cap B(x) = max(\mu A(x), \mu B(x))$, $x \in U$. In the example under consideration, the output membership function looks like in Fig.6 (the shaded area).

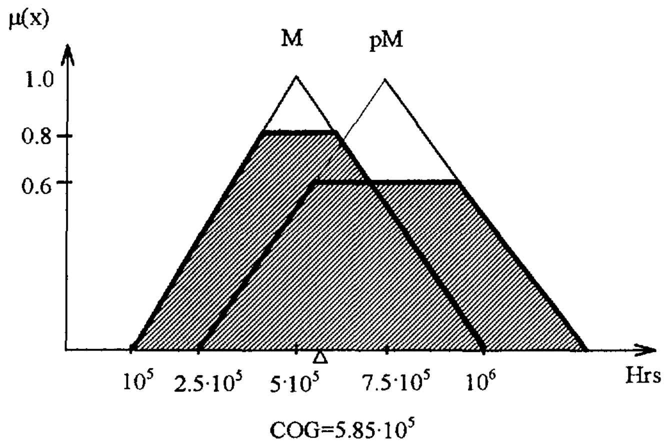


Fig.6. Output membership function of mean-time to failure of metallization

The next step is to find the centre of gravity of the shaded area as

$$\mathrm{COG} = \frac{\int_a^b \mu(x)x\,dx}{\int_a^b \mu(x)\,dx}.$$

More practically an estimation of COG is calculated over a sample of points in the output domain with a step size enough to reach the sufficient accuracy:

$$\mathrm{COG} = \frac{\sum_{x=a}^{b} \mu(x)x}{\sum_{x=a}^{b} \mu(x)}.$$

Frequently used singleton output membership functions considerably simplify the defuzzification procedure. In this case truncated singleton values of output membership functions are combined using a weighted average.

Applying the COG method to the output membership function of mean-time to failure of metallization, the crisp value of 5.85·105 hours is obtained (Fig.6).

2.1.3.Fuzzy control.

The development of the fuzzy logic theory has stimulated alternative ways of solving automatic control problems. Based on ideas of fuzzy logic, Mamdani and Assilian [5] proposed fuzzy controllers which describe human control in a linguistic form. As a result the first applications of the fuzzy control replaced human operators. But it is only recently, since about 1990, that the interest in the fuzzy control has increased strongly, because of successful fuzzy control of industrial processes, success and advertisement of Japanese consumer products. Table 1 [6] that contains the estimations of degrees of sharp and vague information for the main elements in process information: the input variables, the process classes, the automation functions and the output variables, indicates the reasons of fuzzy logic application to control.

According to [6] some advantages of fuzzy control can be summarised as follows:

For processes where no sufficient control performance can be reached with PI- and PID-controllers and parameter adjustment by tuning rules, the fuzzy control may be an alternative. This is especially valid for processes with difficultly understandable behaviour, if only qualitative knowledge is available and strongly non-linear properties exist.

If manual control is possible the operator's knowledge can be formulated in fuzzy rules with linguistic inputs and outputs.

An additional advantage of the fuzzy control concept is a fast prototyping by some few rules, if classical standard controller is not suitable.

Table 1. Degrees of sharpness and vagueness for main elements in process automation. Intensity of a property: +++ large; ++ medium; + small.

Automation elements	Degree of sharp and vague information	
	Crisp	Fuzzy
1.<u>Input Variable</u>	Precise Values	Imprecise Values
- Usual sensor	+++	
- Low cost sensor	+	++
- Non directly measurable variables	++	+
- Linguistic		+++
2.<u>Process Classes</u>	Quantitative Knowledge	Quantitative Knowledge
- Mechanical processes	++	+
- Electrical processes	++	+
- Thermal processes	++	+
- Chemical processes	+	++
- Biological processes	+	++
3.<u>Automation function</u>	Precise Algorithms	Rules
- Control	++	+
- Supervision	+	++
- Management	+	++
- Man-machine interface	+	++
4.<u>Output-Variable</u>	Precise Values	Imprecise Values
- Actuator	+++	
- Operator	+	++

Comprehensive review of different fuzzy control architectures can be found in [6].

2.1.4. Fuzzy logic applications.

Nowadays fuzzy logic algorithms are widely used in microcontrollers, e.g. fuzzy coprocessor SAE 81C99 of Siemens [7]. The coprocessor is able to process within one control cycle up to 256 8-bit inputs and to form up to 64 output control signals using programmable knowledge base containing up to 16384 rules. The coprocessor is manufactured by 1 micron CMOS technology and having frequency 20 MHz is able to process 7.9 million rules per second. The coprocessor is shown in Fig.7.

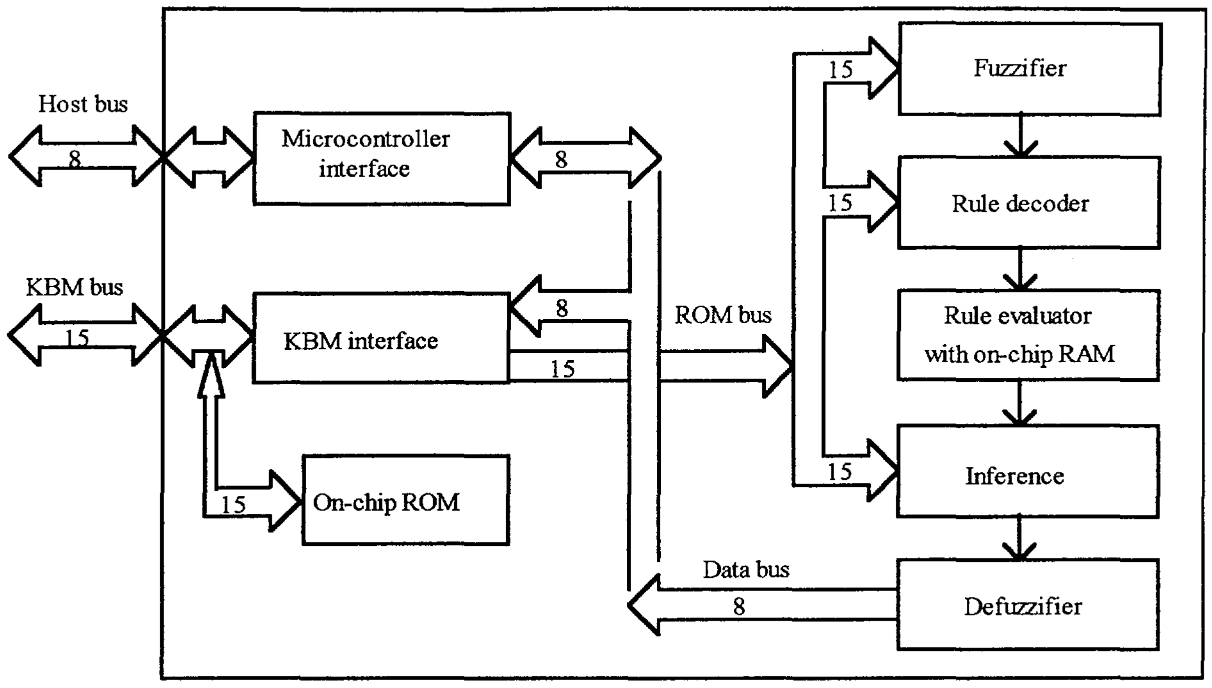


Fig.7. Architecture of Fuzzy Logic Coprocessor

Fuzzy Logic has emerged as a profitable tool for the controlling of subway systems and complex industrial processes, as well as for household and entertainment electronics, diagnosis systems and other expert systems.

Here is a short list of examples of fuzzy logic applications sourced from WWW http://www.mitgmbg.de/erudit:

Automatic control of dam gates for hydroelectric-power plants (Tokyo Electric Pow.).

Preventing unwanted temperature fluctuations in air-conditioning systems (Mitsubishi, Sharp).

Efficient and stable control of car-engines, Cruise-control for automobiles (Nissan, Subaru).

Positioning of wafer-steppers in the production of semiconductors (Canon).

Optimized planning of bus time-tables (Toshiba, Nippon-System, Keihan-Express).

Prediction system for early recognition of earthquakes (Inst. of Seismology Bureau of Metrology, Japan).

Medicine technology: cancer diagnosis (Kawasaki Medical School).

Recognition of handwritten symbols, objects, voice (Sony, Hitachi, Hosai Univ.)

Recognition of motives in pictures with video cameras; Back light control for camcorders; Compensation against vibrations in camcorders (Canon, Minolta, Sanyo, Matsushita).

Automatic motor-control for vacuum cleaners with recognition of surface condition and degree of soiling (Matsushita).

Controlling of machinery speed and temperature for steel-works (Kawasaki Steel, New-Nippon Steel, NKK).

Controlling of subway systems in order to improve driving comfort, precision of halting and power economy (Hitachi).

Improved fuel-consumption for automobiles (NOK, Nippon Denki Tools).

Improved sensitiveness and efficiency for elevator control (Fujitec, Hitachi, Toshiba).

Improved safety for nuclear reactors (Hitachi, Bernard, Nuclear Fuel div.).

Practice of fuzzy logic application reveals that it results in a better performance, a lower power consumption, a higher degree of adaptability, a higher autonomy. Its current state makes it possible to say that the "fuzzy logic industry" really exists.

2.2. Neural Networks

2.2.1.Basic concepts

Many hopes of specialists in artificial intelligence in 50s were dealt with the conception of perceptron developed by F.Rosenblatt [8,9]. But in the end of 60s they were ruined by M.Minsky and S.Papert [10]. Since the end of 70s there has been a growing interest among different specialists to neural models which are its further development.

Neural networks, unlike fuzzy systems, do not require explicit representation of process knowledge in a set of rules. Being a highly interconnected non-linear system, neural nets elicit knowledge (learn) directly from a set of input/output data examples (training data) that represent the process behaviour. This is an important feature which perfectly matches the problem of developing complex systems adaptive to changing input conditions. Moreover, neural networks are very helpful when dealing with incomplete or noisy input signals because they can extract knowledge from incoming data that only partially similar to the training data.

The areas most suitable for neural nets applications are:

pattern recognition (classification);
function estimation;
feature extraction and filtering;
data compression and decompression;
statistical analysis.

Neural networks as one of intelligent technologies were inspired by the brain and human neurones. Neural network models consist of neural elements (Fig. 8) and are based on the activities of processing elements (PE) - "nodes", strengths of connections among the nodes ("weights"), and techniques for adapting the connection strengths in order to improve network performance over time [11].

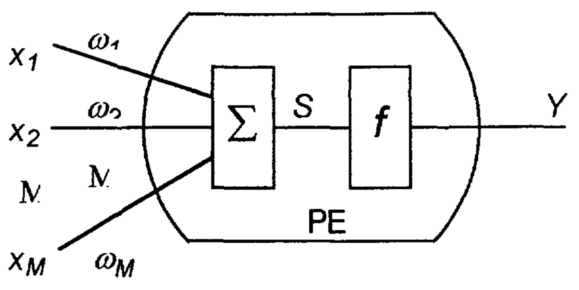


Fig 8. Neural element

A processing element (PE) in a neural net can receive many inputs *x1, x2, ..., xM* (or an input vector *X*), and generates one output *y* that is some non-linear function *f* of its inputs. The operations performed by the processing element depends upon the type of neural network algorithm used. Usually it aggregates its inputs as a weighted sum with weights *w1, w2, ..., wM*: $S = \sum_{i=1}^{M} w_i x_i$, where weights are associated with the interconnections between processing elements and generates an output *y=f(S-Θ)* if this aggregation is above some threshold value *Θ*. Usually the simplest non-linear functions are used: binary function (threshold function, hard limiter) (Fig. 9a) that provides the PE with a digital output

$$y = \begin{cases} 1, & \text{if } S > \Theta \\ 0, & \text{if } S \le \Theta \end{cases}$$

or sigmoid function that provides an analog output (Fig. 9b)

$$y = \frac{1}{\left(1 - e^{-(S-\Theta)}\right)}.$$

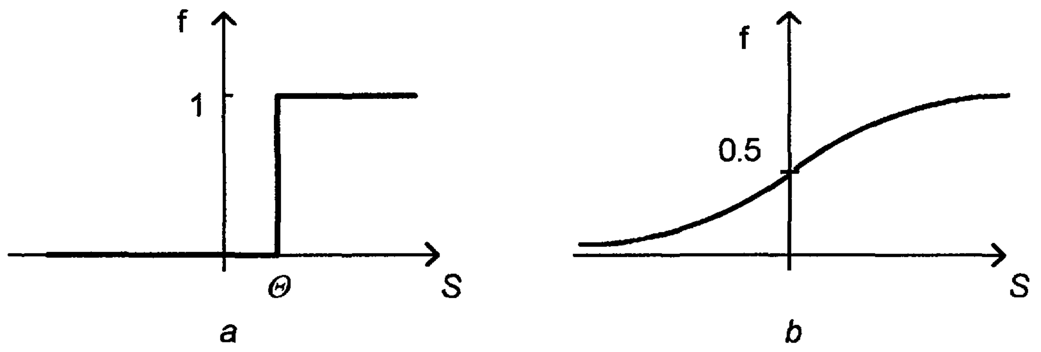


Fig. 9. Tipical non-linear functions

The processing element is said to "fire" if a sufficient excitation is applied to its inputs.

Processing elements are connected into complex arrangements depending on the type of neural network architecture implemented. The most common neural network architecture is the three-layer network (Fig. 10), which usually consists of an input layer (one node for each input variable), an output layer (one node for each output) and a middle or "hidden" layer in which the number of nodes is a configurable parameter that has a significant influence on the performance of the network.

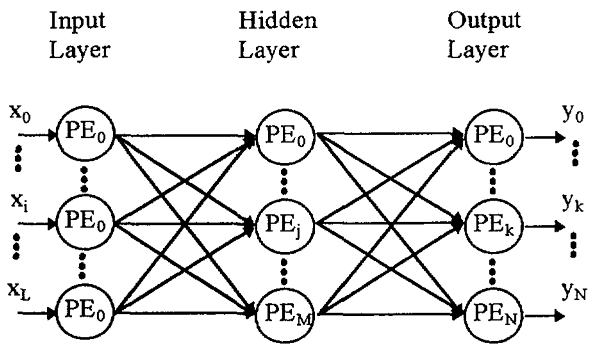


Fig.10. Neural network with three layers.

One of the main features of the neural nets is their selforganization and selfadaptivity. Some training is required before a neural network can be used. During this training process, input data (also called training data) is presented to the network and the connection weights at each PE are adjusted based on a mathematical algorithm called a learning law. Learning can be supervised or unsupervised. In supervised learning the network is presented with inputs and the desired outputs. The weights are then adjusted to minimise the error between the actual output and the desired output. An example of supervised learning is the backpropagation network.

In unsupervised learning only input data is presented to the network. Here, the learning law provides information that allows the network to modify itself based solely on features in the input data. Examples of unsupervised learning are self organising maps and learning vector quantization networks.

The most widely used learning laws are listed below.

Hebb's Rule states that if the connected neurones fire, the connection should be strengthened. This learning rule is the foundation of most learning techniques used in artificial neural networks.

Generalized delta rule. This rule also known as the least mean square (LMS) learning rule since it minimises the mean square error. It is based on the idea of modifying the strengths of connections to reduce the difference between the desired output value and the current output value of a processing element. The backpropagation technique is the most commonly used generalisation of this rule.

Kohonen's learning law. This law is used only in unsupervised learning. In this law, processing elements must compete for the opportunity to learn. The PE with the largest output will be the winner and can then either inhibit its competitors or excite its neighbours. Only the winner can have an output and only the winner and its neighbours can adjust their weights. Since the winning element is defined as the one that has the closest match to the input pattern, Kohonen networks model the distribution in the inputs.

After the training process is finished, and the neural network has learned a proper output response, it can be used in it's "normal" operating mode in the target application.

2.2.2.Neural nets models.

There are many training algorithms which were developed for different neural nets models some of which are described below in more detail.

Multilayer perceptron.

While the simple (one layer) perceptron defines borders of a decision area in a form of hyperspaces, the two layer perceptron can define any convex area in a space of output signals. As for multilayer perceptrons, they are able to realise any input-output mapping. The back-propagation network (BPN), also known as a multilayer perceptron (MPL) network, is one of the most popular neural network algorithms of this type.

The BPN is a multilayer network requiring that at least two of the layers contain adaptive weights. The input layer distributes the inputs to the hidden layer. The network is typically fully connected, that is, every PE in a layer has a connection to every PE in the subsequent layer (Fig. 10).

A processing element in a BPN performs a weighted sum of its inputs: $S = \sum_{i=1}^{M} w_{ii} x_i$,
where:
xi: is the input to the PE, and
wji: is the weight from input *i* to PEj of the following layer.

Then an output function (a transfer function, threshold function or squashing function) is applied to the sum of weights. This non-linear function is used to limit the output of the PE. Otherwise, the potential could have a value significantly larger than the input value. This can cause an explosion in the output value when PEs are connected in a cascade manner.

The BPN requires a training data to be presented. The data sample consists of pairs of patterns between which a relation should be determined. In a multilayer perceptron there can be many hidden layers between input and output ones. Each neural element can be connected to any element of neighbour layers, but not to the elements of the same layer. The training of the BPN includes two phases: a forward-propagation or a processing phase, and a back-propagation or learning phase.

During the processing phase, an input vector is presented to the network. The input layer simply passes the inputs to the first hidden layer (multiple hidden layers are allowed, although a single layer is most common). Each PE in this hidden layer performs the weighted sum of its inputs, applies an output function to this aggregation, and passes the result to the following layer. The process is repeated for all layers in the network until the output layer generates an output.

In terms of pattern recognition, an input vector means a set of characteristics (symptoms), but an output vector corresponds to a class of patterns. Here a hidden layer is used for knowledge representation.

During the learning phase, an output error is computed and then propagated back through the proceeding layers. The weights in the layers are adjusted to minimise the error between the actual output and the desired output of each PE

At the beginning of the learning phase all weights are preset to small random values and a training input vector is presented. Then an output error for each PE in the output layer is computed

$$e_k = (d_k - y_k) \cdot f'\left(\sum_{k=1}^{N} w_{kj} \cdot y_j\right),$$

using previously determined values of outputs yj for every PE in the hidden layer

$$y_j = f\left(\sum_{i=1}^{M} w_{ji} \cdot x_i\right)$$

and the outputs yk for every PE in the output layer

$$y_k = f\left(\sum_{j=1}^{M} w_{kj} \cdot y_j\right)$$

Then this error is propagated back through the proceeding layers by calculating the error for each PE in the hidden layer

$$\tilde{e}_j = f'\left(\sum_{j=0}^{L} w_{ji} \cdot x_i\right) \cdot \sum_{k=1}^{N} e_k \cdot w_{kj} .$$

The weights at the output layer and the hidden layer are updated after each presentation of a training input vector

$$w_{kj}(t+1) = w_{kj}(t) - \xi \ e_k y_j$$
$$w_{ji}(t+1) = w_{ji}(t) + \xi \ \tilde{e}_j x_i .$$

Here the following notations are used:

xi: input *i*.

wji: weight connecting unit *i* in the input layer to PEj in the hidden layer.

yj: output of PEj in the hidden layer.

wkj: weight connecting PEj in the hidden layer to PEk in the output layer.

yk: output of PEk in the output layer.

f(): output function (typically a sigmoid).

ek: error of the output layer.

dk: desired output of unit k in the output layer.

$\tilde{e}_j$: error at the hidden layer.

f'(): derivative of the PE's output function *f*(). *f'*(x)=*f*(x)(1-*f*(x)) for a sigmoid function.

ξ: learning rate (usually a small number between 0.05 and 0.25)

All steps are repeated for all training vectors until the error for all the training vectors is acceptable. With the backpropagation algorithm, multiple presentations of the training set are usually necessary.
In some situations the backpropagation algorithm may not find an optimum solution due to local minima. A local minimum occurs because of the BPN learning law is based on a gradient descent function. There can be other reasons why the BPN may not converge. In this case there is an options to modify any of the design parameters (network architecture: number of layers, number of PEs in a layer, number of inputs/outputs, connectivity between layer bias terms; learning parameters: weight initialisation, learning rate, momentum; training data) and to try the procedure again.

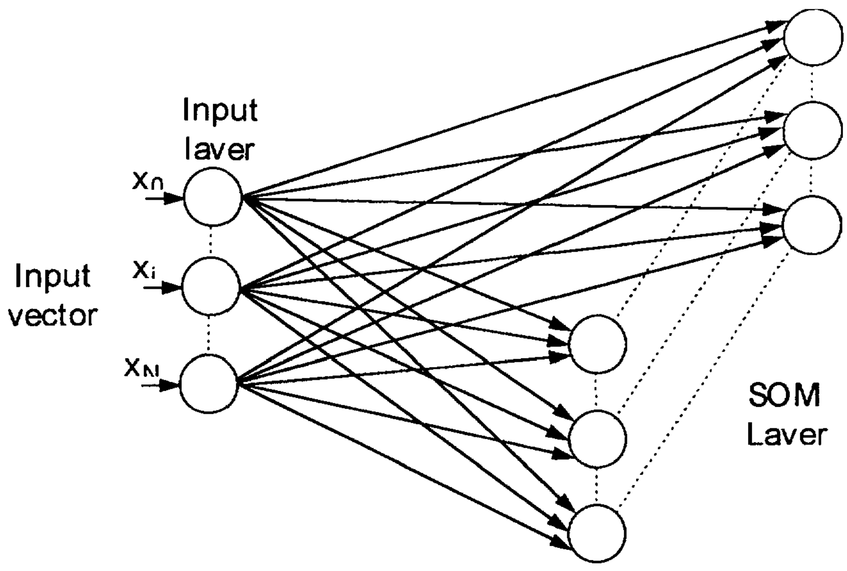
If the network is properly trained, it can be used in its processing phase in the target application. Design flexibility in modifying any number of BPN parameters along with the ability to learn a tremendous variety of complex input-to-output mapping functions from training data make the multilayer perceptron very efficient in different applications

But for the adequate training, the training data consisting of system inputs and proper outputs must be available and should be carefully selected. Moreover the learning algorithm is computationally intensive, and sometimes for it to converge to an optimal solution, the multiple presentation (hundreds) of all training data are required. Anyway this is probably the most popular model of neural network and it is often used for control and optimisation..

Self-Organising Maps

Self-organising maps (SOM), also called feature maps, were first used by T.Kohonen [12,13] for pattern recognition and classification applications. The SOM usually is a two-layer network (Fig.11).

a)



b)

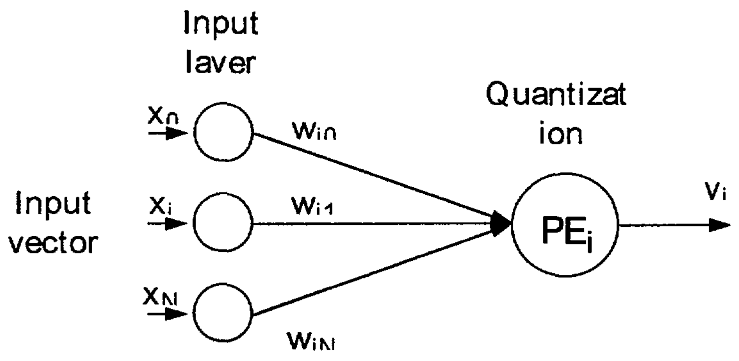


Fig.11. Self-organising map

The input layer is fully connected to the second layer, which is also called the quantization layer, competitive layer or SOM layer. The input layer passes the input values to the SOM layer. The PEs in the SOM layer are sometimes called quantization units and are organised into a two dimensional array. The number of PEs in the input layer determines the dimensionals of the input space. The figure depicts an N-dimensional input layer and the weights associated with quantization unit *j* in the SOM layer (Fig. 11b). The weights connecting to a quantization unit also form an N-dimensional vector *W* which sometimes is called a quantization vector.

SOMs use an unsupervised learning law based on competitive learning. Competitive learning allows the units in a layer to compete amongst each other for the right to adjust their weights. The "winning" unit will then make its weights more similar to the input vector. Training of the SOM allows the weights in the SOM layer to fall into clusters (groups) based on relationships in the input data. The similarities between patterns in the input space are thus mapped into closeness relationships in the weights of the SOM. This has the effect of "projecting" the relationships among patterns in the large-dimensional SOM layer. Patterns in the input space can then be easily visualised in the weights of 2-dimensional SOM layer.

The SOM training algorithm starts with initialising of the weight vector *W* and applying a training vector *X* to its inputs. Training vectors are presented sequentially to the network. When the input vector is received, the distance *dj* between the input vector and each weight vector is computed

$$d_j = \sqrt{\sum_{i=0}^{n}\left(x_i - w_{ji}\right)^2}$$

Then the winning PE in the competitive layer, having the smallest distance *dj,* is selected. The training algorithm then selects the PE that best matches the input vector by finding the weight vector *W* that is the closest (in N-dimensional Euclidean space) to the input vector *X* which is declared the winner. After that the weights of the PEs in the neighbourhood Nc are adjusted

$$w_{ji}(t+1) = w_{ji}(t) + \xi\left(x_i(t_1) - w_{ji}(t)\right),$$

while the weights of other PEs (outside Nc) remain unchanged

$$w_{ji}(t+1) = w_{ji}(t).$$

The training algorithm is repeated for all training vectors.

The following symbols are used:

xi: i-th component of input vector *X*.

wji: weight connecting input unit *X* to PEj in the SOM layer.

dj: distance between an input vector and the weights of PEj in the SOM layer.

η: the learning rate.

Nc: the neighbourhood around the winning PE.

Kohonen recommends that the learning rate η and the neighbourhood size Nc around the winning quantization unit be decreased as the training progresses. Usually, the pattern is to start with a larger definition of the neighbourhood and then narrow it as the training continues.

After training is completed, an input vector presented to the network is classified by the PEs that fire in the SOM layer. The SOM network can thus be used for pattern recognition and classification applications.

Self organising maps are able to classify large and complex input spaces and its fast training algorithm requires just a few presentations of the training set. But sometimes the SOM can lose some important information in low occurrence pattern. These neural networks are good for problems that require classification of faults and other process states. They can directly capture probability distributions, temporal relationships and other important characteristics of manufacturing systems.

The latest neural systems for manufacturing can train themselves to predict process behaviour, using examples obtained directly from on-line process data. All this utility brings only a modest requirement to specify the internal workings of the system - the system itself is largely able to organise the parameters that it needs to classify on-line data, recognise process states, and make predictions. This means that process specialists can implement and manage model-based applications that they never would have attempted using traditional statistical or modelling approaches.

2.2.3.Applications of neural nets.

Now there are a lot of practical realisations of neural systems in different forms: software products for personal computers or workstations [14,15,16], neural coprocessors connected to a personal computer (or workstation) which increase the size of a modelling network and the system productivity.

The strong attention is paid to R&D of hardware for neural systems using modern VLSI technology. Here are some examples of recent VLSI implementations announced at IEEE International Conference on Neural Networks (ICNN'95):

Self-Learning Analog Neural Network LSI with computational power of 10 MCPS and weight updating rate of more then 40 kHz has been fabricated using 1.3-μm double-poly CMOS process [17].

Hardware implementation of multilayer neural network with on-chip learning. Experimental circuit runs with 25 MHz clock, and delivers 14.6x106 CPS and 10.9 x106 CUP [18].

The compact CMOS VLSI design for recursive neural networks with the capability of hardware annealing which can solve optimisation and signal processing problems [19].

Leading electronics companies and research centres (AT&T Bell Labs, Intel, Nestor, JPL, MIT Lincoln Laboratory) also show their significant interest to hardware implementation of neural networks for a wide range of applications. The greatest potential of neural nets is the high speed processing that could be provided through massively parallel VLSI implementation.

The scope of neural system application ranges from pattern recognition and image processing (fully digital VLSI neural system for motion detection from image sequences [20], machine vision for obstacle sensing in cars [21], realisation of high-order CMAC model for colour calibration [22]), supervision and control of technical processes (prediction of end conditions of basic oxygen steelmaking process [23]) to biomedical applications (ECG classification [24], analysis of multivariable medical images [25]) and business (bankruptcy predictions [26]).

Neural networks and fuzzy logic systems can be combined to capture the best of both approaches. So they can be considered as complementary intelligent technologies.

2.3. Genetic Algorithms

2.3.1. Basic concepts

The need to solve optimisation problems is a dominant theme in the engineering world. A great number of analytic and numerical optimisation techniques were developed. Although there are still large classes of functions which present significant difficulties for numerical techniques, and moreover, for analytical methods. These are not continuous or differentiable everywhere, functions which are non-convex, multi-modal, and functions which contain noise. As a consequence, there is a continuing search for more robust optimisation techniques capable to overcome such problems.

Genetic Algorithms have attracted a great deal of attention regarding their potential as optimisation techniques for complex functions. GAs are a heuristic search method based on mechanics of natural selection and natural genetic, a form of stochastic production systems. It combines the survival of the fittest among string structures with a structured yet randomised information exchange to form a search algorithm. While randomised, GAs efficiently exploit historical information to speculate on new search points with expected improved performance[27].

Prior to the use of genetic algorithms for search in artificial systems, a number of biologists used computers to perform simulations of natural genetic of the early 1960s. The first mention of the words "genetic algorithm" and the first published application of a genetic algorithm to game-playing program both came in Bagley's (1967) dissertation. In 1970 Cavicchio in his dissertation "Adaptive Search Using Simulated Evolution" applied genetic-like algorithm to the design of a set of detectors for a pattern-recognising machine. The first application of GAs to a pure problem of mathematical function optimisation was Hollstien's (1971) work "Artificial Genetic Adaptation in Computer Control Systems". Techniques under the name *Evolutionstrategie* were independently developed at the Technical University of Berlin (Rechenberg, 1965).

The present understanding of the term "Genetic Algorithm" came from a milestone work by John Holland "Adaptation in natural and artificial systems" [28] and De Jong's dissertation "An analysis of the behaviour of a class of genetic adaptive systems", as well as Grefenstette's software GENESIS [29]. These works have formed a theoretical base for artificial system designers that retains the important mechanisms of natural systems.

To solve an applied task, GAs maintain a set of strings (population of "chromosomes"). Each string represents an encoded parameter set, a solution alternative, or a point in the search space. Usually, the natural parameter set of the optimisation problem is coded as a finite-length bit string. GAs evaluate a population based on application dependent criteria, and generate a new one iteratively. A new population of chromosomes is generated using probabilistic genetic-like operator, called crossover and mutation, with the intent of generating chromosomes which map to high fitness values. In other words, beneficial changes of parents are combined in their offspring, and the GA adapts to the problem being solved. The commonly used generational model of GA is shown below:

```
**Procedure** GA
**begin**
          Generate initial population, P(0); t=0;
```

```
        Evaluate chromosomes in P(0);
        repeat
            t=t+1;
            Select P(t) from P(t-1);
            Recombine chromosomes in P(t) using genetic operators;
            Evaluate chromosomes in P(t);
        until termination condition satisfied;
    end.
```

GA starts with an initial population of chromosomes chosen at random. By contrast to other search techniques, GAs have no need for auxiliary information about features of search space, they only require the value of an application dependent objective function to be associated with an individual chromosome. Each member of the initial population must be evaluated using this function. Objective function associates a numerical value (also called a fitness value) with a chromosome, which serves as some measure of goodness that is to be maximised. Further steps of GA are repeated iteratively until either all chromosomes have the same associated fitness value (convergence condition) or the desired number of iterations is reached. Each iteration of the algorithm consists of two basic steps: selection and recombination. Selection is a process in which chromosomes are copied to the next population according to their objective function values. The recombination usually proceeds in two steps. First, members of the new population are mated at random. Second, each pair of chromosomes is recombined using a crossover operator. Optionally, mutation operator may be applied to some members of the newly generated population. Finally, each chromosome in the obtained population undergoes the evaluation.

2.3.2. Example of GA application to optimisation problem

Consider the following real parameters optimisation problem:

Minimise $f(x_1,x_2,\ldots,x_N)$,

where each x_i is a real parameter subject to a_i x_i b_i for some constants a_i and b_i .

The objective of the problem is to find the values of $\mathbf{x}=\{x_i\}$ to obtain the minimal possible value of function f. GAs have been successful at solving problems of this type that are too ill-behaved (such as multi-modal and non-differentiable) for more conventional hill-climbing and derivative based techniques.

For example, let the problem under consideration be minimisation of the function $f(x_1,x_2)=x_12+x_22$, which -1$x_1$1 and -2 x_2 2. The first step is to code the parameter vector **x** as a finite-length bit string using either a standard binary coding or a Gray coding. The bit strings for the parameters are concatenated together to give a single bit string (or "chromosome") which represents the entire vector of parameters. Thus, the problem is transformed into a combinatorial problem where the points of the search space are the corners of a high-dimensional cube.

The m-bit binary code for real parameter **x** can be obtained as a binary representation of an integer $k=[(x-a)/(b- a)*2_m]$, where [E] denotes the closest integer less than E. Let there be four bits representing each parameter. Then the point (0.25,1) would be represented by the bit string 1010 1100 using binary coding.

Gray coding is another way of coding parameters into bits which has the property that an increase in one step in the parameter value corresponds to a change of a single bit in the code. The conversion formula from binary coding to the Gray one is:

$$\gamma_k = \begin{Bmatrix} \beta_1 & \text{if } k = 1 \\ \beta_{k-1} \oplus \beta_k & \text{if } k > 1 \end{Bmatrix}$$

where γ_k is the k-th Gray code bit, β_k is the k-th binary code bit, bits are numbered from 1 to N starting from the left, and $\oplus$ denotes addition mod 2. The conversion from Gray coding to binary coding is:

$$\beta_k = \sum_{l=1}^{k} \gamma_l$$

where the summation is done mod 2. So, the binary code 1010 1100 corresponds to the Gray code of 1111 1010.

GA starts with an initial randomly generated population of strings and thereafter generates successive populations of strings. Let the initial population of the size N=4 be the following: (1010 1100), (1101 0101), (0011 0110), (0100 1011), where the binary coding is used. To evaluate the initial population, it is necessary to decode each chromosome and to apply objective function to it. The result of evaluation is shown in Table 2.

Table 2 Initial population of chromosomes

No.	Chromosomes	**x**=(x1,x2)	Fitness value f(**x**) = x12 + x22
1	1001 0010	(0.125,-1.5)	2.27
2	1110 0101	(0.75,-0.75)	1.125
3	0110 1100	(-0.25,1.0)	1.0625
4	1011 1111	(0.375,1.75)	3.203

The selection operator may be implemented in a number of ways. The most popular one is based on a biased roulette wheel, where each slot corresponds to a chromosome in the population and its size is proportional to the chromosome's goodness. The corresponding weights of roulette slots pi can be calculated according to the following formulas:

$$\text{pi} = \frac{f_i'}{\sum_{k=1}^{N} f_k'}, \quad f_k' = \frac{\sum_{j=1}^{N} f_j}{f_k}$$

, where fj - the fitness value of j-th chromosome.

Each spin of the roulette wheel selects an i-th chromosome with the probability pi. Spinning roulette N times, N candidates are determined for a new population subjected to further genetic operators. Let p1=0.17, p2=0.34, p3=0.36, p4=0.13. Suppose that values 0.15, 0.3, 0.5, 0.7 were obtained as a result of four roulette spins. The value 0.15 belongs to the first sector of the roulette between 0.0 and 0.17; values 0.3 and 0.5 belong to the second sector (0.17,0.51); and the value 0.7 belongs to the fourth sector (0.51,0.87). Thus chromosomes with numbers 1,2,2,3 are selected. The result of the selection is summed up in the Table 3.

Table 3 Result of the selection

No.	Fitness f(**x**)	f'(**x**)	Expected probability pi	Count from roulette wheel
1	2.27	3.37	0.17	1
2	1.13	6.81	0.34	2
3	1.06	7.21	0.36	1
4	3.20	2.39	0.13	0
Total:	7.66	19.78		
Average:	1.91			

After selection, a recombination operator is to be applied. Members of the new population are mated at random, let it be the following pairs: (1,2) and (3,4). Then each pair of strings undergoes crossing over with the crossover site selected at random, see Table 3. According to a simple crossover algorithm, two new strings are created by swapping substrings starting from the crossover site. For example, consider strings 1 and 2 in the population obtained after selection and crossover site is 2:

10 | 01 0010

11 | 10 0101

As a result of applying simple crossover two strings of a new population are obtained, as follows:

10 | 10 0101

11 | 01 0010

The result of recombination: the new population decoded **x**=(x_1,x_2) and the corresponding fitness values are given in Table 4.

Table 4 Results of the recombination

No.	Population after reproduction	Crossover site	New population	Decoded $\mathbf{x}=(x_1,x_2)$	New fitness $f(\mathbf{x})$
1	10|01 0010	2	1010 0101	(0.25,-0.75)	0.625
2	11|10 0101	2	1101 0010	(0.625,-1.5)	2.640
3	1110 01|01	6	1110 0100	(0.75,-1.0)	1.560
4	0110 11|00	6	0110 1101	(-0.25,1.25)	1.625
				Total:	6.450
				Average:	1.610

The population average fitness has improved from 1.915 to 1.61 in one generation. The minimum fitness has decreased from 1.0625 to 0.625. This improvement is no fluke. The best string of the first generation id copied twice because of its high performance. When it is crossed with the other one, the offspring inherit best properties of their parents.

The process repeated iteratively for a new population until GA termination condition is satisfied, i.e. when the desired number of iteration is reached or when all strings in the population become identical (convergence condition). The best string in the final population represents the best solution of the problem reached at the moment of GA termination.

2.3.3. GA's operators

The genetic operators have a key position in GAs, they are so chosen that manipulating the population leads GA away from unpromising areas of the search space and towards prospective ones, without the GA having to explicitly remember its trail through the search space.

The crossover operator plays a central role in the GAs, and, in fact, could be considered to be one of the algorithm's defining characteristics. Numerous investigations have been aimed at finding powerful crossover operators. For a finite-length string representation of chromosomes, crossover operator produces new strings by exchanging substrings from the parents. The number of crossover points usually fixed at a very low constant value of 1 or 2. This decision came from the early theoretical works and recommended to use a 2-point crossover. However, there are situations in which having a higher number of crossover points is beneficial [30]. Perhaps the most interesting and effective operator (uniform crossover) was introduced by Syswerda [31]. In the uniform crossover the allele of any position in an offspring is copied from the first parent with probability P0 and from the second parent with probability 1-P0.

For a wide variety of problem domains, such as Travelling Salesman Problem (TSP) and Job Scheduling Problem (JSP), it is natural to code solutions as a permutation of integers which represent a sequence of visited cities or a sequence of scheduling operations. The traditional recombination operators for bit string representation of solutions do not work when the solutions are coded as sequences. Obviously, the string that represents a tour of cities must contain exactly one instance of each city label, while GAs traditionally assume that the symbols within a string can be independently modified and rearranged. Therefore, a number of crossover operators were suggested for problems of this class, i.e. for permutation of integers[32].

Consider, for example, the Order Crossover which creates two offspring preserving the order and the position of symbols in a subsequence of one parent while keeping the relative order of the remaining symbols of the other parent. Let there be two parents selected for recombination:

Parent1:	123|456|78
Parent2:	735|128|46

Two crossover points are chosen at random. The symbols between crossover points are copied from the Parent1 into same positions of the Child1. Then, starting just after the second crossover point, the symbols are copied from the Parent2, omitting symbols that were copied from the Parent1. When the last symbol of Parent2 is reached, the process continues with the first one in the Parent2 until all of the symbols have been copied to the child. The second child is formed in the same way by switching the roles of the parents.

Child1:	128|456|73
Child2:	456|128|73

Mutation operator is used for finding new points in the search space to evaluate. Suppose a population containing strings which have j-th bit equal to 1. Crossover operator can not change that bit to 0 because it operates with substrings presented in the population and it preserve position of any substring. Thus, mutation could introduce some new features to population. When mutation is applied to a binary coded chromosome, a random choice is made of some of its bits and they are flipped from 0 to 1 or from 1 to 0. Normally the mutation rate is very low, so that it would be unlikely to have for a string more than one bit mutated. For the permutation of integers, mutation can be applied by swapping two integers at arbitrary chosen positions.

2.3.4. Analysis of GAs behaviour

Holland has suggested [28] that the better way to view GAs is in terms of optimising a sequential decision process involving uncertainty in the form of lack of a priory knowledge, noisy feedback, and time-varying payoff functions. Holland showed via his Schemata Theorem that, given a number of assumptions, GA is quite robust in producing near optimal sequences of trials for problems with high level of uncertainty.

While using GA for solving optimisation problems there is a tendency to treat GA as a tool for function optimisation. However, it is not true. The proper understanding of GAs behaviour helps to understand their potential application to any particular problem. Several common viewpoints on GA behaviour were clearly pointed out by De Jong.

The genotypic viewpoint is to consider the contents of the population as a gene collection and study proportions of gene value (allele) over time. It is quite clear that, if fitness proportional selection is the only mechanism driving evolution of GA, an initial population rapidly evolves to a population containing only duplicate copies of the best individual in the initial population. Genetic recombination operators counterbalance this selective pressure by providing diversity in the form of new alleles and new combinations of alleles. When recombination operators are applied at fixed rates, the population evolves to some point of dynamic equilibrium at a particular diversity level. Most of the individuals of this population are identical and represent optimal or near optimal solution of the problem under consideration.

A phenotypic viewpoint is to focus on physical meaning of the genes. In the optimisation example discussed above, the phenotype is the set of pairs of reals (x_1,x_2) defined by binary strings. Plotting over time positions of the population members on the fitness landscape, it is possible to observe that the population rapidly shifts to a region close to some point. Frequently, this is an optimal point, but not always, especially for multi-modal fitness surfaces.

The optimisation point of view focuses on monitoring the average fitness or the best fitness value of individuals. The plot of the best individual fitness over time looks like "optimisation curve", it shows that fitness value become more and more close to some "optimal" value. Measurements of that sort have encouraged the view of GA as a tool for function optimisation. The genotypic and phenotypic viewpoints show that the population does not, in general, converge to a steady state containing multiple copies of the global optimum. So, comparing GA with other optimisation methods is possible only in terms of obtaining the best decision as a function of the amount of effort involved (the number of trials).

2.3.5. Applications of GAs

GAs have been successfully tried on NP-hard combinatorial optimisation problems, such as network link optimisation and travelling salesperson problem; and they are of interest in themselves as a primitive model for natural selection. Also GAs have been applied to such problems as design of semiconductor layout and factory control, and used in AI systems and neuromorphic networks to model processes of cognition, e.g. language processing and induction. Here are some examples of GAs application:

Multiobjective Optimisation [33].
Fault coverage test code generation [34].
VLSI circuit layout and compacting via GA [35,36].
Scheduling multiprocessor tasks [37]
Assembly-line balancing problem using GA [38].
Adaptation of robot behaviour [39].
Transportation problem [40].
Design of system reliability [41].
Designing telecommunication networks [42,43].
Management of Isochronous Channels Reusing in LANs [44].

GAs are theoretically and empirically proven to provide robust search in complex spaces. Many examples state the validity of the technique in function optimisation, control applications, VLSI design, and machine learning. Having been developed as a valid approach to the problems requiring efficient and effective search, GAs are now finding a more widespread application in business, scientific and engineering circles. The reason for the growing number of applications is that GAs are computationally simple yet powerful in their search for improvement. Furthermore, they are not fundamentally limited by restrictive assumptions about the search space (those concerning continuity, existence of derivatives, unimodality, and others). Unlike other search algorithms, the probabilistic primitives that GAs use to manipulate their populations, are very fast on contemporary hardware [45].

2.4. Rough set theory

Rough set theory is a relatively new mathematical and intelligent technique introduced in early 1980's by Pawlak [46]. The technique is particularly suited to reasoning about imprecise or incomplete data, and discovering relationships in this data. The primary application of the rough sets so far has been in data and decision analysis, databases, knowledge based systems, and machine learning. The main advantage of the rough set theory is that it does not require any preliminary or additional information about data.

In some sense the rough set theory exploits the same approach as the fuzzy set theory - revision of the classical set theory. The rough set philosophy is based on the assumption that, in contrast with the classical set theory, there is some additional information (knowledge, data) about elements of the universe. But if the fuzzy logic allows gradual and continuous transition of logic or qualitative expressions such as "high", the rough set theory, on the other hand, identifies degrees of significance for input and output attributes and, by means of "quantization", discerns vague information. The rough control also allows descriptive or qualitative expressions.

The basic idea of the rough set theory is in the assumption that elements that display the same information, are indiscernible in terms of the available information and form elementary granules (blocks, classes, etc.) of knowledge. These granules are called elementary sets or concepts and are considered as elementary building blocks of our knowledge. Elementary concepts can be combined into compound concepts, i.e. concepts that are uniquely defined in terms of elementary concepts. Any union of elementary sets is called a crisp set, and any other sets are referred to as rough (vague, imprecise).

Within the rough set theory the two crisp sets called the lower and the upper approximation of *X*, are associated with every set *X*. The lower approximation of *X* is a union of all elementary sets which are included in *X*, whereas the upper approximation of *X* is a union of all elementary sets which have non-empty intersection with *X*. In other words, the lower approximation of a set is the set of all elements that surely belong to *X*, whereas the upper approximation of *X* is a set of all elements that possibly belong to *X*. The difference of the upper and the lower approximation of *X* is its boundary region. Obviously, a set is rough if it has a non-empty boundary region; otherwise the set is crisp. Elements of the boundary region cannot be classified, employing the available knowledge, either to the set or to its components. Approximations of sets are basic operations in the rough set theory and are used as main tools to deal with vague and uncertain data.

The proposed method of data analysis has several advantages. Some of them are listed below:

1. Provides efficient algorithms for finding hidden patterns in data.
2. Finds minimal sets of data (data reduction).
3. Evaluates significance of data.
4. Generates minimal sets of decision rules from data.
5. It is easy to understand and offers straightforward interpretation of results.

Precise mathematical formulations of the above ideas can be found in [46].

3. Promising future

Many industrial problems can be solved by combining components of the “Soft Computing” in different ways. In the solving process they complement rather than compete with each other. Thus, fuzzy logic and neural networks are successfully used in pattern recognition, manufacturing processes, control, etc.

Although, Fuzzy Logic was invented in the United States the rapid growth of this technology had started from Japan and had now again reached the USA and Europe as well. Fuzzy Logic is still booming in Japan, the number of patent applied for, increases exponentially. Fuzzy has become a key-word for marketing. Electronic articles with intelligent component have more opportunities for market success.

In Japan the Fuzzy-research is widely supported with a huge budget. In Europe and the USA efforts are being made to catch up with the tremendous Japanese success. For instance, the NASA space agency is engaged in applying Fuzzy Logic for complex docking-maneuvers.

In order to support the development of Intelligent Technologies and to disseminate knowledge and expertise within the European Union, between universities and industry companies, ERUDIT - the European Network of Excellence in ”Uncertainty Techniques Developments for Use in Information Technology” - has been started by European Commission (DG III Industry - ESPRIT programme) in 1995.

Summing up, the following main tendencies characterise the current state of application of Soft

Computing in the modern industry:

- Leading microelectronics companies (Siemens, Motorola, Texas Instruments) launched manufacturing of electronic devices implementing intelligent technologies and successfully created the respective market;
- Growing interest and support of R&D in intelligent technologies from governments and international organisations;
- Expansion of activities from pure academic research towards development of industrial applications.

These factors combined together form a powerful vehicle to drive the former extremely exotic and mostly academic science to promising future of real life applications.

References

1. L.A.Zadeh. *Foreword by the Honorary Chairman.* In Proceedings of the Second European Congress on Intelligent Techniques and Soft Computing EUFIT'94, Aachen, Germany, September 20-23, 1994.
2. Zadeh L. *Fuzzy sets.* Information Science. - 1965. - No8, p.338.
3. Zimmermann H-J. *Fuzzy set theory and its applications.* - Kluwer Academic Publishers, Boston, MA, 1992.
4. Zadeh L. *Outline of a new approach to the analysis of complex systems and decision process.* IEEE Trans. on SMC. - 1973. - Vol. 3, No.1, p.28-44.
5. Mamdani E., Assilian S., *An experiment in linguistic systhesis with a fuzzy logic controller.* International Journal of Man-Machine-Studies, Vol 7, pp.1-13.
6. Isermann R. *On fuzzy logic applications for automatic control, supervision and fault diagnosis.* - Proceedings of the Third European Congress on Intelligent Tehniques and Soft

Computing. EUFIT'96 Aachen, Germany - August 28-31, 1995, p.738-753.

7. *Fuzzy Coprocessor SAE 81C99.* Data sheet, SIEMENS AG, Semiconductor Group, München, April, 1994.

8. Rosenblatt F. *The perceptron: A probabilistic model for information storage and organisation in the brain.* Psychol. Rev., Vol. 65, p.386-408.

9. Rosenblatt F. *Principles of neurodynamics.* Baltimore, 1962.

10. Minsky M., Papert S. *Perceptrons.* Cambridge, 1969.

11. S. Fraleigh. *Fuzzy Logic and Neural Networks. Practical Tools for Process Management.* PC AI, May/June 1994, pp.17-21.

12. Kohonen T. *Self-organized formation of topologically correct feature maps.* Biol. Cybern. - 1982, Vol. 43, p.59-69.

13. Kohonen T. *Self-organization and associative memory.* Berlin, Springer, 1984.

14. P.Wilke, J.Rehder, G.Billing, J.Nilson, C.Mansfeld. *NeuroGraph - An Integrated Development Environment for Neural Networks, Genetic Algorithms and Fuzzy Logic.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.1879-1883.

15. D.Georges, B.Geropp, A.Seeliger. DiDa - *A Tool for the Training of Neural Networks.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.1884-1887.

16. J.Angstenberger. DataEngine - *A Software Product Family for Intelligent Data Analysis.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.1892-1896.

17. Morie T., Fujita O., Uchimura K. *Self-Learning Neural Network LSI with High-Resolution Non-Voltage Analog Memory.* In Proceedings of the 1995 IEEE International Conference on Neural Networks (ICNN'95), Perth, Western Australia, pp. 1628-1631.

18. Hikawa H. *Implementation of Simplified Multilayer Neural Networks with On-chip Learning.*

In Proceedings of the 1995 IEEE International Conference on Neural Networks (ICNN'95), Perth, Western Australia, pp. 1633-1637.

19. Chou E.Y., Sheu B.J., Jen S.H. *A Compact VLSI Design for Recursive Neural Networks with Hardware Annealing Capability.* In Proceedings of the 1995 IEEE International Conference on Neural Networks (ICNN'95), Perth, Western Australia, pp. 1650-1655.
20. A.Abruzzo, M.Gioiello, M. La Cascia, F.Sorbello. *Motion Detection from Image Sequences Using a New Fully Digital VLSI Neural Architecture.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.1341-1345.
21. M.Brauckmann, C.Goerick, T.Zielke. *Automatic Visual Sensing for Cars.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.1396-1402.
22. Ker J-S., Kuo Y-H., Liu B-D *Hardware realization of High-order CMAC Model for Color Calibration.* In Proceedings of the 1995 IEEE International Conference on Neural Networks (ICNN'95), Perth, Western Australia, pp. 1656-1661.
23. Keuthen M., Seeliger A., Thornton S. *Neural Net for Process Condition Prediction in Steelmaking.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.304-309.
24. C.Ramirez-Rodriguez, T.Vladimirova. *A Hierarchical Fuzzy Neural System for ECG Classification.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.1668-1672.
25. F.Firenze, A.Schenone, F.Acquarone, M.Gambaro, F.Masulli. *NeuroGraph - An Adaptive Resolution Analysis of Multivariable Medical Images via Unsupervised Neural NetworkBased Clustering.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.1690-1694.
26. P.Protzel, J.Wallrafen, H.Popp, J.Baetge. *Bankruptcy Prediction Using Different Soft*

Computing Methods. In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.1710-1714.

27. Goldberg D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
28. Holland J.H. *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press, 1975.
29. Grefenstette GENESIS: A system for using genetic search procedures.- Proceedings of the 1984 Conference on Intelligent Systems and Machines, 161-165.
30. Spears W.M., De Jong K.A. *An Analysis of Multi-Point Crossover.*- In Foundations of Genetic Algorithms, Ed. G. Rawlins, Morgan Kaufman, San Mateo, California, 1991, pp.301-315.
31. Syswerda G. *Uniform Crossover in Genetic Algorithms.*- In proceedings of the 3rd International Conference on Genetic Algorithms, 1989.
32. Fox B.R., McMahon M.B. *Genetic Operators for Sequencing Problems.*- In Foundations of Genetic Algorithms, Ed. G. Rawlins, Morgan Kaufman, San Mateo, California, 1991, pp.284-300.
33. Voget S. *Multiobjective optimization with Genetic Algorithms and Fuzzy-Control.*- In Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing EUFIT'96, Aachen, Germany, September 2-5, 1996, pp.391-394.
34. Bilchev G., Parmee I., Darlow A. *The Inductive Genetic Algorithm with application to the fault coverage test code generation problem.*- In Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing EUFIT'96, Aachen, Germany, September 2-5, 1996, pp.452-456.
35. Davis L., Smith D. *Adaptive design for layout synthesis* (Texas Instruments internal report). Dallas: Texas Instruments, 1985.

36. Fourman M.P. *Compacting of symbolic layout using genetic algorithms.* Proceeding of an International Conference on Genetic Algorithms and their Applications, 141-153, 1985.

37. Sze-Sing Lam, Cai X.,Lee C.-Y.. *A Genetic Algorithm for scheduling multiprocessor tasks without prespecified processor allocation.*- In Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing EUFIT'96, Aachen, Germany, September 2-5, 1996, pp.448-451.

38. Tsujimura Y., Gen M., Li Y., Kubota E. *An. Efficient method for solving fuzzy assembly-line balancing problem using GA.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.406-415.

39. Leitch D., Probert P. *A Fuzzy Model of the Evolution of Behaviors in Robotics using Genetic Algorithms* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.447-452.

40. Ida K., Gen M., Li Y. *Solving multicriteria solid transportation problem with fuzzy numbers by Genetic Algorithms.* In Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing EUFIT'95, Aachen, Germany, August 28-31, 1995, pp.434-441.

41. Gen M., Yokota T., Taguchi T. *Optimal design of system reliability using Genetic Algorithm.* Proc. of 10th Fuzzy Symp., Osaka, Japan, 1994.

42. Abuali F.N., Schoenefeld D.A., Wainwright R.L. *Designing Telecommunications Networks using Genetic Algorithms and Probabilistic Minimum Spanning Trees* Proceedings of the 1994 ACM Symposium on Applied Computing (SAC'94) March 6-8, 1994, Phoenix, AZ, 242-246.

43. Abuali F.N., Schoenefeld D.A., Wainwright R.L. *Terminal Assignment in a Communications Network using Genetic Algorithms* Proceedings of the 22nd Annual Computer Science Conference, March 8-10, 1994, Phoenix, AZ, 74-81.

44. Vasilakos A., Markaki M., Kassotakis I. *A Hybrid Genetic Algorithm for an effective management of Isochronous Channels in LANs/MANs.*- In Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing EUFIT'96, Aachen, Germany, September 2-5, 1996, pp.438-442.

45. Rawlins G. J. E., editor. *Foundations of Genetic Algorithms,* San Mateo, 1991. Morgan Kaufmann.

46. Z.Pawlak, *Rough sets: Theoretical aspects and reasoning about data,* Dordrecht, Netherlands: Kluwer Academic, 1991.