Synthesis of Intelligent Hybrid Systems for Modeling and Control

by

Wael Farag

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical Engineering

Waterloo, Ontario, Canada, 1998

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

The intelligent information processing performed in humans is now being mimicked in a new generation of adaptive machines as the *state-of-the-art* technology. Inspired by the functionality of brain nerve cells, artificial neural networks can learn to recognize complex patterns and functions, and based on the biological principle of "survival of the fittest", genetic algorithms are developed as powerful optimization and search techniques. Likewise, fuzzy logic imitates the mechanism of approximate reasoning performed in the human mind, and hence can reason with linguistic and imprecise information.

Although these intelligent techniques have produced promising results in some applications, certain complex problems cannot be solved using only a single technique. Each technique has particular computational features (e.g. ability to learn, explanation of decisions) that make it suitable for particular problems and not for others. These limitations have motivated the creation of *intelligent hybrid systems* where two or more techniques are combined. Although there is an increasing interest in the integration of fuzzy logic, neural networks, and genetic algorithms to build *intelligent hybrid systems*, no systematic synthesis framework has been developed so far. Therefore, the objective of this thesis is to construct an *intelligent learning scheme* that incorporates the merits and overcomes the limitations of the three paradigms. The applications considered for the proposed scheme are modeling and control.

The generic topology of the system used in this thesis has a transparent structure; its parameters, links, signals and modules have their own physical interpretations. Moreover, the learning scheme uses task decomposition to identify the systems' parameters. The learning task is decomposed into three subtasks (phases). The first phase performs a coarse identification for the systems' numerical parameters using unsupervised learning (clustering) algorithms. The second phase finds the linguistic-association parameters (linguistic rules) using unsupervised as well as supervised learning algorithms. In the third phase, the numerical parameters are optimized and fine-tuned using supervised learning and search techniques. The performance of the scheme is assessed by testing it on two benchmark modeling applications. The results are compared to that of other intelligent modeling approaches to show the performance characteristics of the proposed scheme. The scheme is also assessed by applying it to nonlinear control problems. The synchronous machine voltage regulation and speed stabilization problems have been tackled using the proposed scheme. Several comparative studies are carried out to show the advantages of the proposed control approach over conventional approaches.

Acknowledgments

First and foremost, I gratefully thank and praise *Allah* the almighty, the only One God, for enlightening my way and directing me through each and every success I have ever reached or may reach.

I would like to thank Prof. *Victor H. Quintana* for his guidance, supervision, and financial support. Also, I would like to thank Prof. *Germano L. Torres* for his assistance, vision, and time he devoted to this work.

I wish to express my gratitude to Dr. *Ahmed Tawfik* for his friendly assistance, helpful comments and invaluable time.

I'm really indebted to my wife *Amany Wahba*, for the unlimited support, understanding and encouragement throughout the past four years; and to my son *Ahmed* for his innocent smiles that filled my life with pleasure and happiness.

My sincere thanks to my brother, sister, and friends for their constant encouragement and care.

To my parents, *Abdel-moniem Farag* and *Fayza Khamis*, I would like to extend my profound thanks and appreciation for their everlasting love, deep understanding, permanent patience, and moral support.

For the generation which will fulfill the prophecy
of spreading true peace on earth.
And for those who are striving
to bring this generation up.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Preface

Humans are hybrid information processing machines. Our actions are governed by a combination of genetic information acquired through learning. Information in our genes hold successful survival methods that have been tried and tested over millions of years of evolution. Human learning consists of a variety of complex processes that use information acquired from interactions with the environment. It is the combination of these different types of information processing methods that has enabled humans to succeed in complex, rapidly changing environments.

This type of *hybrid* information processing is now being mimicked in a new generation of adaptive machines as the *state-of-the-art* technology. The applications range from aircraft control systems that diagnose and repair themselves to systems that can successfully trade in foreign exchange markets [1-8]. At the heart of these adaptive machines are *intelligent* computing *systems*, some of which are inspired by the mechanics of nature.

Neural Networks (NNs), for example, are inspired by functionality of nerve cells in the brain. Like humans, neural networks can learn to recognize patterns by repeated exposure to many different examples. They are good at recognizing complex patterns such as hand-written characters and financial markets decisions.

Genetic Algorithms (GAs), are also naturally inspired and based on the biological principle of "survival of the fittest". The main idea behind a genetic algorithm is the evolution of a problem's solution over many generations, with each generation having a better solution than its predecessor.

While these intelligent techniques have produced encouraging results in particular tasks, certain complex problems cannot be solved by a single *intelligent* technique alone. Each intelligent technique has particular computational properties (e.g. ability to learn, explanation of decisions) that make them suitable for particular problems and not for others. For example, while neural networks are good at recognizing complex patterns, they are not good at explaining how they reach their decisions. Fuzzy logic systems, which can reason with imprecise information, also have particular strengths and limitations. They are good at explaining their decisions but they cannot automatically acquire the rules they use to make those decisions (lack of learning ability). These limitations have been a central driving force behind the creation of *intelligent hybrid systems* where two or more techniques are combined in a manner that overcomes the limitations of individual techniques [9].

1.2 Problem

A leading application domain for *intelligent hybrid systems* is process modeling and control. Control systems based on fuzzy logic and neural networks are no longer just research topics. Indeed, their popularity has redefined the field of *Intelligent control*, in which much activity is now devoted to the investigation of hybrid architectures that integrate neural networks, fuzzy logic, genetic algorithms and other novel (or newly resurgent) technologies.

Since fuzzy logic approach has been proposed by *Zadeh* [10-11], fuzzy modeling and control are considered one of the most attractive strategies in tackling complex control and decision systems. Fuzzy logic strategies are particularly suitable for nonlinear systems with imprecise and/or uncertain knowledge of their parameters and behavior. Fuzzy control systems, unlike conventional control systems, have a large number of parameters to be tuned during the design process. However, most of the current implementations and designs of fuzzy models/controllers rely mainly on a substantial amount of heuristic observations to express the system strategy's knowledge and tune its parameters [9, 12]. Therefore, the practical development of such systems still suffers from two critical problems: finding the system-strategy's initial rules, and tuning the initial rules and their membership functions. Moreover, it is difficult for human experts to examine all input-output data recorded from a complex process to find, and tune the rules and their membership functions within fuzzy systems. Therefore, a fuzzy logic system should be integrated (or augmented) with some techniques that can provide learning, adaptation and optimization capabilities to this system.

Neural networks and genetic algorithms are promising learning synergisms to be integrated with fuzzy logic in order to construct a suitable *intelligent hybrid system* for modeling and control applications. This integration enables the system to handle both quantitative and qualitative knowledge. In other words, the system can be learned from the available input-output properties (data) as well as the designer experience.

1.3 Objectives

Although there is an increasing interest in the integration of fuzzy logic, neural networks, and genetic algorithms to build *intelligent systems* for modeling and control applications, such integration of these three technologies (according to our knowledge) can't be found in the literature so far. However, there is already significant literature in the integration of neural networks or genetic algorithms with fuzzy logic for modeling and control. Hence, one of the objectives of the current research is to construct an intelligent learning scheme that incorporates the merits of the three paradigms with application focused on modeling and control.

Another objective is to overcome the limitations of each approach by compensating such limitations with some of the salient features in other approaches. For example, neural networks have limitations in handling qualitative knowledge and they are black box approaches which their actions or decisions can't be expressed by natural language. This limitation is well compensated by the linguistic reasoning provided by fuzzy logic. On the other hand, fuzzy logic lacks the capability for learning and adaptation which is compensated by the powerful learning and adaptation capabilities in neural networks. Genetic algorithms, when integrated with fuzzy logic and neural networks, grant this *hybrid system* with robust search and optimization technique that reduces the dependency on the designer experience for synthesizing a model or controller.

One of the main advantages behind the popularity of fuzzy control is that it is a model-free approach with high capability of reasoning under nonlinearity and uncertainty. This advantage is preserved in the *intelligent hybrid system* proposed in this thesis. The proposed synthesis approach does not depend on *a priori* quantitative model of the plant under study or need to evaluate any parameter in this plant. However, the input-output properties (data) of the plant under study is mainly required to perform the learning/adaptation/optimization tasks in this approach. This does not present an obstacle as such data, in most cases, is usually available or easy to be collected.

Also, an invaluable objective in the proposed approach is to utilize and incorporate all the available sources of information about the plant in the synthesis process. These

sources could take the form of observations and/or experience, numerical data or model, and linguistic data. The synthesis as well as the topology of the proposed *hybrid system* are organized in a manner that can satisfy this objective.

1.4 Approach

In order to synthesize an *intelligent hybrid system*, two requirements have to be specified. The first requirement is the topology (structure) which shows the distribution of the different parameters of the system and how they interact together. The second requirement is the learning scheme which adapts the system parameters according to the received information from the surrounding environment.

In this thesis, the selected topology of the system is transparent. In other words, all its parameters, links, signals, and modules have their own physical interpretations (or meanings). Unlike most neural networks structures, the structure used does not have a black box form, instead it has a *glass box* form. This important feature grants us a better understanding of the system's reaction and behavior.

The number of parameters which should be found in order to synthesize the intelligent systems, are usually large. Also, the parameters themselves take different forms such as numerical numbers or linguistic associations (like linguistic rules). Identifying all these parameters instantaneously is a very difficult task and found to be impractical. Thus, it is more convenient to divide the learning scheme into subtasks. In this thesis, the proposed learning scheme is divided into three phases. The first phase performs a coarse identification for the systems' numerical parameters using unsupervised learning (clustering) algorithms. The second phase is used to find the linguistic-association parameters (linguistic rules) using unsupervised as well as supervised learning algorithms. In the third phase, the numerical parameters are optimized and fine-tuned using supervised learning and search techniques.

Numerical data as well as expert knowledge could be incorporated together in all the three learning phases. However, the utilization of the expert knowledge, in the second learning phase, is more vital and effective. The selection of the numerical training data has also a remarkable effect on the overall design specially in control applications. Thus, the incorporation of expert knowledge in choosing the appropriate training data is also effective.

1.5 Thesis overview

The dissertation spans nine chapters. Chapter 2 presents a review of the fundamental concepts of fuzzy control, neuro-control, and neuro-fuzzy control with an extensive survey of the work done in these areas. Moreover, an overview on genetic algorithms and the work emphasizing the application of them in the optimization of fuzzy and neuro-controllers are presented.

In Chapter 3, the proposed *intelligent hybrid learning* scheme as well as the suggested topology are described. Also, the associated task decomposition and the suggested candidates for each subtask are discussed in details.

In Chapter 4, the coarse identification learning phase is described. The suggested techniques are also presented in detail. A comparative study among them is carried out using different evaluating indices.

In Chapter 5, the linguistic-rule formation phase is described, the proposed algorithms are presented, and a comparative study among them using a well-known benchmark is carried out.

In Chapter 6, the optimization learning phase is described, the proposed algorithms are also presented, and a comparative study using a well-known benchmark is carried out. Moreover, a sensitivity analysis is performed to investigate the effect of some parameters on the algorithms used in this learning phase.

In Chapter 7, the application of the proposed *hybrid learning scheme* in modeling of complex dynamical systems is presented. Two well-known benchmarks are used to show the effectiveness of the proposed scheme.

In Chapter 8, the application of the proposed *hybrid learning scheme* in nonlinear control problems is presented. The proposed scheme is used to design different control schemes for synchronous machines. A review of the state-of-the-art applications of artificial intelligence techniques in synchronous-machine control is presented. The synchronous-machine voltage regulation and speed stabilization problems are tackled. Many comparative studies are carried out to show the advantages of the proposed control approaches over conventional approaches.

In Chapter 9, the contribution of this dissertation is emphasized. Conclusions and suggested future work are also presented.

Chapter 2

Background and Literature Survey

2.1 Introduction

In recent years it has been recognized that to realize more flexible control systems it is necessary to incorporate other elements, such as logic, reasoning and heuristics into the more algorithmic techniques provided by conventional control theory [13], and such systems have come to be known as intelligent control systems. The technical committee on intelligent control of the IEEE Control Systems Society has defined the general characteristics of intelligent control as having an ability to emulate human capabilities, such as planning, learning and adaptation [14-15]. Learning and adaptation especially are essential characteristics of intelligent control systems, and while adaptation does not necessarily require a learning ability for systems to be able to cope with a wide variety of unexpected changes and environments, learning is invariably required.

It is necessary to specify the characteristics that qualify a system to be justifiably recognized as an intelligent control system. First and foremost, intelligent control systems are designed to maintain satisfactory closed-loop system performance and integrity over a wide range of operating conditions. The characteristics of the system must therefore relate to the complexity of the plant, including non-linear and time-variant plant behavior, dimensional and other multivariable characteristics, the complexity of the desired performance objective, imperfection and uncertainties in the measurements, and an ability to cope with component failures.

In recent years the use of the terminology *'intelligent control'* has come to embrace diverse methodologies combining conventional control theory and emergent techniques based on physiological metaphors, such as neural networks, fuzzy logic, artificial

intelligence, genetic algorithms and a wide variety of search and optimization techniques. This chapter reviews the aspects of these emergent techniques, namely, fuzzy logic, neural networks and genetic algorithms that pertain to the realization of intelligent control systems. The fundamental concepts of each paradigm are also discussed.

2.2 Fuzzy logic control

Conceptually, fuzzy logic control systems are rule-based expert systems which comprise if .... then (condition/action) rules of the form [11]

$$\begin{aligned} Rule^r\text{: } IF \quad & X_1 \text{ is } A_1^r \text{ and } X_2 \text{ is } A_2^r \ldots X_n \text{ is } A_n^r \\ THEN \quad & Y_1 \text{ is } B_1^r \text{ and } Y_2 \text{ is } B_2^r \ldots Y_m \text{ is } B_m^r \end{aligned} \tag{2.1}$$

where $X_i \in U_p$ and $Y_j \in V_q$ are process input and output variables, and A_i^r and B_j^r are their actual values for the rth rule, respectively. These values generally depict linguistic (or fuzzy) terms, such as high, cold, negative which are represented by fuzzy sets defined on the corresponding universes of discourse U_p and V_q. It is common to use the membership function μ_A of the fuzzy set A, to represent the set itself. One of the most popular membership descriptions is the Gaussian function, which is defined as

$$f_i(x) = e^{-\left(\frac{\|x-m_i\|}{\sigma_i}\right)^2} \tag{2.2}$$

where m_i is the centroid of the fuzzy set (membership function) and σ_i its width. This type of membership functions is characterized by only two parameters, m and σ.

Each fuzzy 'if ... then' rule defines a fuzzy hyperset given by the Cartesian product of the fuzzy sets of the variables in the rule, i.e. $\mu_R = \mu_{A1}{}^r \times \mu_{A2}{}^r \times \ldots \times \mu_{An}{}^r \rightarrow \mu_{Bj}{}^r$. The interpretation and membership functions of these rules depend on the t-norm operator used for the implication of the rule [16]. Typically, the '*min*' and '*algebraic product*' operations are used and, respectively, interpreted as

- min

$$\mu_{A \rightarrow B} = min(\mu_A, \mu_B) \tag{2.3}$$

- algebraic product

$$\mu_{A \rightarrow B} = \mu_A \times \mu_B \tag{2.4}$$

where $\mu_{A \rightarrow B}$ is the resultant confidence factor of a fuzzy rule.

The most common fuzzy inference method is the so called superstar (°) composition rule [11, 17-18] which can be described as follows. Let $A = \{A_1, A_2 \ldots A_n\} \in U$ be the inputs to the fuzzy system. The membership function of the jth output fuzzy set from the rth rule is given by

$$\mu_{B_j}^r(y) = \mu_{(A^\circ B)} = sup(\mu_A(x)^*\mu_R) \tag{2.5}$$

where * is a t-norm operator such as 'min' or 'algebraic' product, and $\mu_{B_j}^r(y)$ is the resultant membership function of the jth output of the rth rule after applying the fuzzy inference. The overall result from a set of R fuzzy rules is an aggregation of the μ_R fuzzy sets, performed a number of ways using t-conorms, such as union or algebraic sum. To obtain a nonfuzzy output from these rules, one of several different defuzzification techniques is applied [17-18]. For example, using the Mean-Of-Centers (MOC) method the output can be obtained as

$$b_j = \frac{\sum_{r=1}^{R} \overline{B}_j^r \mu_{B_j^r}}{\sum_{r=1}^{R} \mu_{B_j^r}} \tag{2.6}$$

where $\overline{B}_j^r$ is the central support of the consequence fuzzy set of each rule and $\mu_{B_j^r}$ is the weight of each rule determined from equation (2.5). It is clear that employing different operators for inference or aggregation can result in different behavior of the fuzzy system. The most commonly used t-norms (t-conorms) are min (max) and algebraic product (algebraic sum).

An alternative approach of fuzzy inference, which was meant to overcome the fact that the supstar inference results in an output which is a fuzzy set rather than a real-valued variable, hence, requiring the use of defuzzifiers, is introduced by *Takagi-Sugeno* [19]. The *Takagi-Sugeno* fuzzy controller uses fuzzy rules which have an 'if' part similar to *Zadeh* [11] rules, but whose 'then' part is a nonfuzzy quantity expressed as a polynomial. The 'then' part is composed as a linear function of 'if' part variables. Rules are typically of the form:

$$\begin{aligned} Rule^r\text{: } IF \quad & X_1 \text{ is } A_1^r \text{ and } X_2 \text{ is } A_2^r \ldots X_n \text{ is } A_n^r \\ & THEN \quad Y = B^r = b_o^r + b_1^r x_1 + b_2^r x_2 + \ldots b_n^r x_n \end{aligned} \tag{2.7}$$

where b_i and x_j are scalar quantities. The overall output from all the rules is obtained as weighted average of the rule outputs

$$\bar{b} = \frac{\sum_{r=1}^{R} w^r B^r}{\sum_{r=1}^{R} w^r} \tag{2.8}$$

where w^r is the truth value of each rule and is calculated as a product of the membership grades of the antecedents

$$w^r = \prod_{i=1}^{n} \mu_{A_i^r(x)} \tag{2.9}$$

Notice that this technique has similarity with the *Zadeh* approach if algebraic product and mean-of-centers are used for inference and defuzzification. The *Takagi-Sugeno* approach provides a compact form suitable for application of parameter-estimation methods. On the other hand, it excludes the possibility for incorporating human expert knowledge.

Figure 2.1 shows a general configuration for implementing a fuzzy logic control system. The system has five distinguishable components: a fuzzifier for converting inputs to fuzzified values in the universe of discourse, a knowledge base containing all the information about fuzzy membership functions of the input variables, a rule base consisting of fuzzy control rules, an inference logic such as pattern matching or supstar composition, and a defuzzifier such as the weighted averaging technique, center of gravity method, and mean of maxima method.

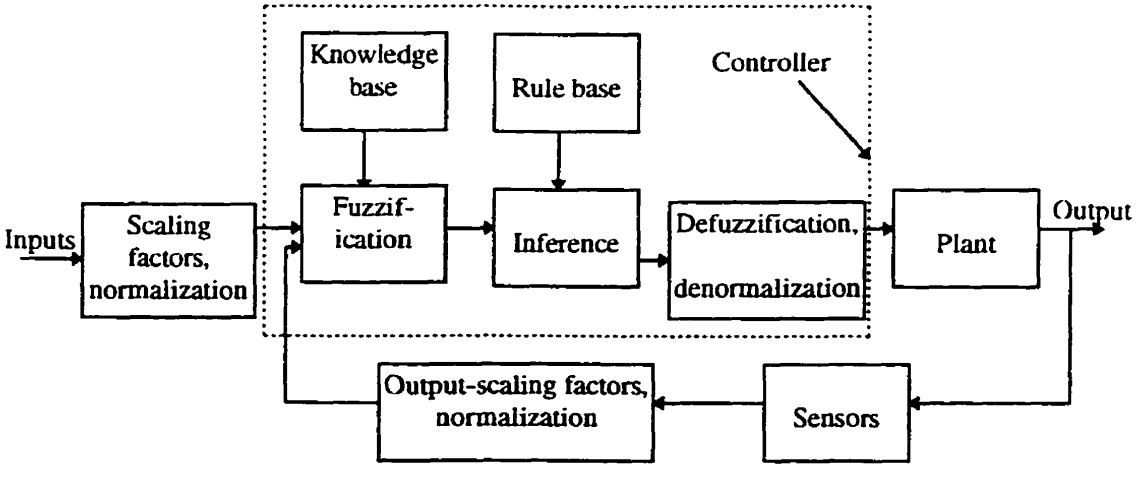


Figure 2.1 A simple fuzzy logic control system block diagram.

Lee [17-18] has presented a survey on different fuzzy controllers design techniques. In particular, the survey includes a discussion of fuzzification and defuzzification strategies, the derivation of the database and control rules, the definition of fuzzy implication, and an analysis of fuzzy reasoning mechanisms. A general methodology for constructing a fuzzy controller has been also described together with the assessment of its performance.

2.3 Neural networks applications in control

Artificial neural networks (ANNs) are biologically inspired and represent a major extension of computation [20]. They embody computational paradigms, based on a biological metaphor, to mimic the computations of the brain. The improved understanding of the functioning of the neuron and the pattern of its interconnections has enabled researchers to produce the necessary mathematical models for testing their theories and developing practical applications. The basic element of any neural network structure is called 'neuron' which is an extremely simple processing element that has multiple inputs and a single output. An input to a neuron could be from the surrounding environment or from other neurons. Also, a neuron output could be fed into other neurons or directly into the surrounding environment. The output of a neuron is constructed by taking the weighted sum of its inputs transformed by a transfer function. Neural networks gain their overall processing capability by connecting these simple neurons to other neurons with an associated weight, which determines the structure of the signal that is transmitted from a neuron to another. The total collection of weights are the parameters that completely specify the model of the process which this net represents. Therefore, in order to learn or identify this model one needs a systematic strategy for adjusting these weights which is called the learning algorithm. In order for the net to learn, one needs to present a number of examples to the net whose attributes are known and are representatives for the unknown model. This set of given examples is called the training set.

The use of neural networks in control systems can be seen as natural step in the evolution of control methodology to meet new challenges [21]. Looking back, the evolution in the control area has been fueled by three major needs: the need to deal with increasingly complex systems, the need to accomplish increasingly demanding design requirements, and the need to attain these requirements with less precise advanced knowledge of the plant and its environment - that is, the need to control under increased uncertainty. Today, the need to control, in a better way, increasingly complex dynamical systems [22] under significant uncertainty has led to re-evaluation

of the conventional control methods, and it has made the need for new methods quite apparent [21-26].

Extensive research has been done in the application of neural networks in control systems [27]. In efforts to provide a consistent classification for the large number of diverse applications, *Werbos* [28] suggested five categories based on network functional approaches. These are: supervised control, where NNs are trained on a database of correct signals; direct inverse control, where NNs learn the mapping between a desirable response trajectory and the control signals that produce it; neural adaptive control where NNs are used in place of standard techniques of model-based adaptive control such as identification of dynamical models; dynamic optimization; and adaptive critic control such as reinforcement learning. These categories are described as follows:

1. *Supervised control:* The training of neural controllers based on human-expert experience is sometimes the only feasible design method, especially for controllers of complex and poorly defined processes for which no suitable conventional controller exists, but for which human experts often make reasonable control decisions based on experience and intuition. Similarly, the neural network may be trained on the actions provided by a conventional controller. The performance of the neural controller in both cases will be limited by the performance of the original controller, and hence this approach is only advantageous in situations where the original controller is computationally more expensive, such as in a process with fast dynamics.

2. *Direct inverse control:* A common application technique is to train the neuro-controller using process open-loop input-output data so that it is able to extract the inverse mapping between the output and control input. Supervised learning is used to learn the mapping between the process state signals and control signals as shown in Figure 2.2. Subsequently, during the operation stage the network is provided with the desired values of the plant output and infers a suitable control signal, and hence used directly as the controller. Direct inverse control is based on the assumption that there exists a one-to-one mapping from the input state to the output state, in which case there also exists an inverse map from the output state to the input state. The plant must, however, be open-loop stable. The inverse map is learned by randomly traversing the input space and building up a database of input-output pairs. Problems are encountered if the mapping from control inputs to plant output is not invertible, or is not one-to-one.

3. *Neural adaptive control:* Adaptive control techniques can be categorized roughly into two classes according to the means by which the controller parameters are adjusted. First is the direct approach such as model reference control where controller parameters are continuously adjusted using, for example, an error-gradient based method to minimize an error between the plant and the reference model. The second approach is the indirect or self-tuning method where first a model of the plant is identified, and then the parameters of this model are used to design the controller. Both the controller and the model may be represented by several networks. In predictive control using NN, a model is used to predict future values of the controlled variable over a horizon of interest, which may be operated in parallel with the process to derive actual on-line plant parameters.

4. *Dynamic optimization:* In this approach, the NN is trained to discover and optimize a control strategy, without guidance of examples or training patterns. The training procedure is, in most cases, carried out on a model of the process; however, schemes where this can be conducted on the plant itself, are significantly more important since the performance of the trained neural controller will be bounded by the accuracy of the model used as shown in Figure 2.3. The training procedure involves assuming a set of parameters for a given neural controller and evaluating a performance measure for the set. A performance measure is used as a training signal. The inputs to the controller are, in general, current and past values of the plant inputs and outputs. Supervised learning techniques can be extended to achieve a self-learning controller using multiple networks [29]. For example, one network is used as an emulator and learns the system dynamics while another is used to control the emulator by minimizing the error between the desired output and the output of the emulator.

5. *Adaptive critic control:* The adaptive critic family of designs is more complex than the other four. One of the most popular such critic designs is the reinforcement learning scheme. Reinforcement control schemes are minimally supervised learning algorithms; the only information that is made available is whether or not a particular set of control actions has been successful. The original application attempted to balance an inverted pendulum, subject to the constraints that the platform should not move more than a certain distance from its starting point and that the inverted pendulum remained approximately upright. If either of these constraints was violated, a failure signal was sent to the learning algorithms. The solution proposed by *Batero et al.* [30] was to construct a control scheme that was composed of two adaptive elements; an Associative Search Element (ASE) and an Adaptive Critic Element (ACE). The ASE attempts to reproduce the optimal control signal that satisfies the given performance objectives, while the

ACE attempts to monitor the performance of the controller internally and to provide an internal reinforcement signal which is used to train the ASE, as illustrated in Figure 2.4.

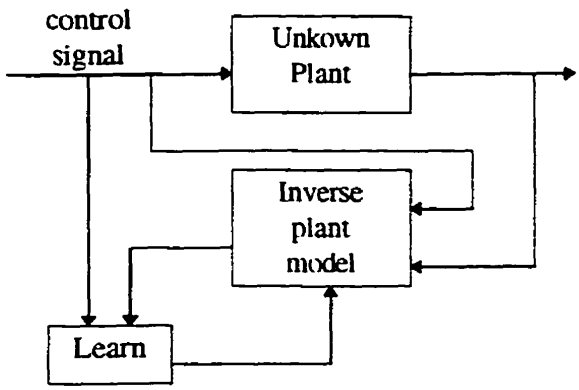


Figure 2.2 Direct inverse plant modeling.

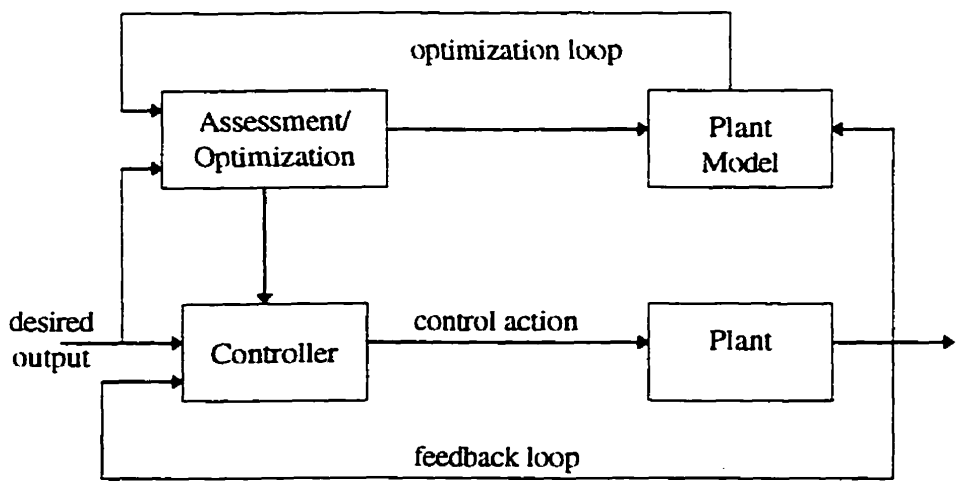


Figure 2.3 Dynamic optimization control architecture.

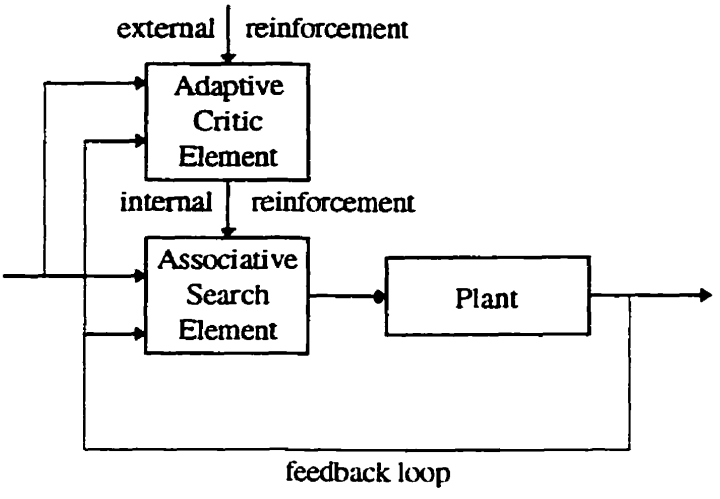


Figure 2.4 An ASE/ACE reinforcement system's architecture.

2.4 Genetic algorithms overview

Genetic algorithms (GAs) are powerful search and optimization algorithms based on the mechanics of natural selection and natural genetics. GAs can be characterized by the following features [31-34].

- A scheme for encoding solutions to the problem, referred to as chromosomes or strings.
- An evaluation function (referred to as a fitness function) that rates each chromosome relative to the others in the current set of chromosomes (referred to as a population).
- An initialization procedure for a population of chromosomes (strings).
- A set of operators which are used to manipulate the genetic composition of the population (such as recombination, mutation, crossover, etc.).
- A set of control parameters that provide initial settings for the algorithm and its operators. Also, the algorithm terminating condition should be specified.

A candidate solution (in a GA) for a specific problem is called a chromosome and consists of a linear list of genes, where each gene can assume a finite number of values (alleles). A population consists of a finite number of chromosomes. The genetic algorithm evaluates a population and generates a new one iteratively, with each successive population referred to as a generation. Given an initial population *P(0)*, the

GA generates a new generation *P(t)* based on the previous generation *P(t-1)* as follows [34]:

Initialize P(t)→P(0) *:P(t) Population at time t*
Evaluate P(0)
While (not terminate-condition) do
  begin
    t← t+1 *:Increment generation*
    select P(t) from P(t-1)
    recombine P(t) *:apply genetic operators (crossover, mutation)*
    evaluate P(t)
  end
end.

The GA uses three basic operators to manipulate the genetic composition of a population: reproduction, crossover, and mutation. Reproduction is a process by which the most highly rated chromosomes in the current generation are reproduced in the new generation. Crossover operator provides a mechanism for chromosomes to mix and match attributes through random processes. For example, if two chromosomes (parents) are selected at random (such as $[a_1\ b_1\ c_1\ d_1\ e_1]$ and $[a_2\ b_2\ c_2\ d_2\ e_2]$) and an arbitrary crossover site is selected (such as '3'), then the resulting two chromosomes (offspring) will be $[a_1\ b_1\ c_1\ d_2\ e_2]$ and $[a_2\ b_2\ c_2\ d_1\ e_1]$ after the crossover operation takes place. Mutation is a random alteration of some gene values in a chromosome. Every gene in each chromosome is a candidate for mutation, and its selection is determined by the mutation probability.

In order to understand how genetic algorithms work we have to introduce the *Holland's schema* theory [35]. A schema is a similarity template among different chromosomes. A schema (over the binary space without loss of generality) is a string of type $(a_1, a_2, ..., a_i, ..., a_n)$, $a_i \in \{0, 1, *\}$. The '*' symbol is a don't care symbol which accepts both '1' and '0'. A schema is a template that describes a sub-space of strings that match the schema at all loci where the schema is specific (specifies either '1' or '0'), and regardless of the value the strings exhibit at the loci of the '*' symbol.

The mathematical basis of *Holland's* schema theorem arises from the observation that in evaluating the fitness of a chromosome one also derives implicit knowledge about the schemata which describe that chromosome. The accuracy of this extrapolation depends on the specificity of the given schema. At the micro-level of a GA, the search is viewed through the space of chromosomes. However, the essence of the schema theorem is that one can also view the changing population as a search

through the set of schemata which the chromosomes instantiate. Since each chromosome is an instantiation of 2^n possible schemata, in testing a chromosome one derives a great deal of implicit information regarding the 'fitness' of the schemata it belongs to. *Holland* calls this *"implicit parallelism"*, and this observation is a major part of the explanation of the power of the GA search.

With the schema theorem in hand, the essence of the chromosome structure in GA optimization becomes clearer. By selecting chromosomes from a sampled population with a probability relative to their fitnesses, one selects representatives of a particular schemata proportionate to their average fitness. The average fitness of a schemata is an artificial quantity that only indicates which chromosome templates are more promising to investigate, and by how much more.

The frequency $m(H, t)$ of a schema H at generation t, will change at generation time $t+1$ proportionally to the respective selection probability for reproduction. More precisely, the growth of a schema due to the proportional replication is given by [33]

$$m(H, t+1) = m(H, t)\frac{\overline{f(H)}}{\overline{f}} \tag{2.10}$$

where the numerator $\overline{f(H)}$ is the average fitness of all chromosomes belonging to the schema H. Similarly, the denominator $\overline{f}$ is the average fitness of the entire population.

Schemata may be disrupted due to crossover (unless, of course, the crossover is performed between identical chromosomes), and therefore, the expected growth of a schema of equation (2.10) is disrupted accordingly. If P_c is the probability for a crossover, and $\delta(H)$ is the metric distance between the first and the last specific schema positions, then the probability of a schema to be disrupted due to crossover is given by [33]

$$P_c\frac{\delta(H)}{n-1} \tag{2.11}$$

A schema can also be disrupted due to mutation. If the probability of a mutation is P_m and the number of specific positions contained in the schema is denoted by the order of the schema $o(H)$, then the probability of a schema being disrupted by a mutation is given by

$$o(H)\,P_m \tag{2.12}$$

and the approximated growth of a schema under crossover and mutation is

$$m(H, t+1) \geq m(H, t)\frac{\overline{f(H)}}{\bar{f}}\left[1 - P_c\frac{\delta(H)}{n-1} - o(H)P_m\right] \qquad (2.13)$$

The schema theorem explains why GAs exhibit high efficiency in search spaces that contain structurally similar sub-spaces, *i.e.* similarities that can be associated with characteristic performance. Thus, by observing the similarities between chromosomes one can advance the search with great efficiency resulting from the implicit parallelism.

As powerful optimization and search techniques, genetic algorithms are used in control systems design by applying them in off-line tuning of controller parameters. For example, Section 2.6 shows how fuzzy controllers could be augmented with genetic algorithms to optimize their parameters and find their rules.

2.5 Fuzzy-neural networks in control systems

There is a rapidly growing interest in the fusion of fuzzy systems and neural networks to obtain the advantages of both methods while avoiding their individual drawbacks. The possibility of integration of these two paradigms has given rise to a rapidly emerging field of fuzzy neural networks. Fuzzy neural networks have become an area of great activity in control engineering and many important problems have been successfully addressed.

There are two distinctive approaches for fuzzy-neural integration. On the one hand, many paradigms that have been proposed simply view a fuzzy-neural system as any ordinary multilayered feedforward neural network which is designed to approximate a fuzzy control algorithm [36-37]. On the other hand, there are those approaches which aim to realize the process of fuzzy reasoning and inference through the structure of a connectionist network [38-40]. Fuzzy-neural networks are, in general, neural networks whose nodes have 'localized fields' which can be compared with fuzzy rules and whose connection weights can similarly be equated to input or output membership functions. The majority of reported studies on fuzzy-neural control applications address one of the following functions:

- using neural networks to tune fuzzy systems [41];
- extracting fuzzy rules from given numeric data examples [37, 40];

- developing hybrid systems combining neural networks and fuzzy systems in various implementation forms [42].

The simplest attempt in merging of fuzzy logic and neural controllers is to make the NN learn the input-output characteristics of a fuzzy controller [43-44]. The NN in this case imitates the fuzzy controller but the only advantage is that the trained NN output has more smoothing robust actions than that of the fuzzy controller.

Evidently, the most common trend has been to apply neural networks to tune the membership functions for defined sets of rules. *Horikawa et al.* [45], for instance, start with a fixed number of rules whose membership functions are subsequently perturbed through backpropagation until they fit a given data.

A new approach that is rapidly gaining interest is to create special architectures out of standard feedforward networks that can be interpreted as fuzzy controllers [39, 40, 42]. The membership functions and sets of rules are constructed from data examples using multi-step procedures that involve learning the membership functions, forming rule representations and constructing computational networks. From the input to the output, these networks are constructed to replicate the structure of a fuzzy controller using either multiple layers or separate networks [40, 46]. *Lin and Lee* [40], for example, proposed a general neural-network connectionist that performs the fuzzy control actions. The proposed fuzzy control/decision network can be constructed from training examples using a multi-step learning scheme. The first step is used to find the initial membership functions using *Kohonen* self-organizing feature maps [47], the second step is used to find the fuzzy rules using a competitive learning technique, and the third step is used to optimize the input/output membership functions using a backpropagation algorithm. Another example is the work of *Hung* [12], in which he proposed the application of *Kohonen* self organizing feature maps and learning vector quantization algorithms for the creation of two-stage training networks for generating fuzzy principles rules from experimental input/output data. The previous examples assume the availability of sufficient training data but in the case of the difficulty of obtaining such training data, reinforcement learning techniques have been proposed and successfully used [42, 48].

Neural networks have also been designed to generate rules autonomously by self-learning methods. *Nie and Linkens* [49, 50] have studied an approach that uses an iterative learning technique to reduce the error between the controlled plant and a reference model, by repeatedly operating the plant through a reference-input profile. The reference model is used to provide desirable plant outputs which guide the modification of the neural networks weights until the error between the actual plant

and the reference model is sufficiently reduced with the number of iterations of plant operation. *Kyung and Lee* [51] have proposed an approach which relies on on-line input-output characteristics of a network-based fuzzy controller to form its rules. Called a quasi-fuzzy logic controller (QFLC), its functional blocks emulate the components of the traditional fuzzy controller, consisting of an inference network and a defuzzification network. Learning is achieved through both reinforcement learning and on-line backpropagation.

Jang and Sun [52] have reviewed the fundamental and the advanced developments in neuro-fuzzy synergism for modeling and control. They formalized the adaptive networks as a universal representation for any parameterized models. The fuzzy model under the framework of adaptive networks is called Adaptive-Network-based Fuzzy Inference System (ANFIS). They have introduced the design methods for ANFIS in both modeling and control applications.

2.6 Fuzzy-genetic applications in control systems

A common difficulty in fuzzy systems is the need for their parameters to be specified by human designers. Following their successful application to a variety of learning and optimization problems, GAs have been proposed as a learning method that can enable automatic generation of optimal parameters for fuzzy controllers, based on an objective criteria. Fusion of fuzzy systems and genetic algorithms has recently attracted interest and a number of successful applications have been reported. There are three application approaches for this fusion. In one case, linguistic fuzzy rules of a conventional fuzzy controller are fixed and their membership functions are optimized [53]. In the second case, the membership functions of specified linguistic values are fixed and the GA is used to determine an optimal set of rules for the application [54, 55]. There is a third approach which combines both and in which rules and membership functions are adjusted simultaneously [56-59]. For example, *Homaifar and McCormick* [59] have proposed a genetic algorithm to tune the fuzzy rules and their membership functions simultaneously; however, they have introduced many assumptions to reduce the number of parameters (search apace) and decrease the convergence computation time.

The interdependence between the linguistic variables and their membership functions suggested that both of them should be adjusted to achieve superior performance. Thus, the membership functions are adjusted individually for each rule and it has been shown that this approach can achieve better results than the other methods. However, one drawback with this type of application is that the ability to

interpret and explain the behavior of the fuzzy controller may subsequently be lost, since the membership functions are no longer associated with only one linguistic name.

Genetic algorithms in general, and also in combination with fuzzy logic, have been very successful in other off-line optimization problems [60], including robot motion planning [33, 61], pH concentration control [53], and shop floor job scheduling problems [62]. GAs have also been applied to fuzzy pattern-classification from numerical data. *Ishibuchi et al.* [63], for example, have proposed a GA to select the most appropriate fuzzy rules from a large set of available fuzzy rules in order to construct a minimal compact set of rules that maximizes the number of correct classifications.

Genetic algorithms, and for that matter any stochastic search techniques, are not suited to real-time control problems because they take a long time to converge and may also inflict severe consequences on the process if produced unpredicted results.

2.7 Neuro-genetic algorithms

Genetic algorithms and artificial neural networks both are techniques for learning and optimization which have been adopted from biological systems. They are both self-learning methods but they use quite different approaches. Neural networks use inductive learning and in general require examples, while GA use deductive learning and require an objective evaluation (fitness) function. A synergism between the two techniques has been recognized which can be applied to enhance each technique performance in what may be referred to as evolutionary neural networks. This is a very recent field and hence there are very few studies. *Schaffer et al.* [64] present a survey of studies on combination of GAs and NNs.

An area that attracted the most interest is the use of GAs as an alternative learning technique in place of gradient-descent methods, such as, error backpropagation. Supervised learning algorithms suffer from a possibility of getting trapped on sub-optimal solutions. GAs enable the learning process to escape from entrapment in local minima in instances where the backpropagation algorithm converges prematurely.

Many studies, on the other hand, have attempted to take advantage of both techniques in hybrid networks. For example, algorithms which combines GA and backpropagation have been shown to exhibit better convergence properties than the pure backpropagation [65, 66]. The GA is used to rapidly locate the region of optimal performance and then gradient descent backpropagation can be applied in this region.

GAs have also been studied as a generalized structure/parameter learning in neural networks. This type of learning combines, as complimentary tools, both inductive learning through synaptic weight adjustment and deductive learning through the modification of the network topology to obtain automatic adaptation of system knowledge to the domain environment [67]. Such hybrid systems are capable of finding both weights and the architecture of a neural network, including the number of layers, the number of processing elements per layer and the connectivity between processing elements [68, 69]. This approach has also been extended to fuzzy neural networks [70, 71]. *Cliff et al.* [72] presented theoretical studies of the use of a genetic algorithm to dynamically evolve neural networks that reflect the complexity of the environment.

2.8 Summary

Over recent years, several techniques have been proposed in the literature for implementing integrated fuzzy-neural, fuzzy-genetic, and neuro-genetic control systems. Many of these have been reviewed in this chapter. However, there is no available approach that integrates the three paradigms (fuzzy logic, neural networks, and genetic algorithms) in one system to capture the merits in each paradigm and to avoid their individual limitations. It has been shown, from the discussion in this chapter, that there are great opportunities for their future integration. Therefore, the main purpose of the next chapter is to present how this integration is achieved and illustrate the advantages and new features that the new hybrid system has.

Chapter 3

Synthesis of Hybrid Systems

3.1 Linguistic modeling

The principle of incompatibility, formulated by *L. Zadeh* [11], explains the inadequacy of traditional quantitative techniques when used to describe complex and *humanistic* systems. *Zadeh* has suggested a linguistic (qualitative) analysis for these systems in place of the conventional quantitative analysis. Accordingly, linguistic modeling of complex systems has become one of the most important issues [73-76]. A linguistic model is a knowledge-based representation of a system; its rules and input/output variables are described in a linguistic form which can be easily understood and handled by a human operator; in other words, this kind of representation of information in linguistic models imitates the mechanism of approximate reasoning performed in the human mind.

The fuzzy set theory formulated by *Zadeh* [10] has been considered an appropriate representation method for linguistic terms and human concepts. *Mamdani's* pioneering work in fuzzy control [3] has motivated many researchers to pursue their research in fuzzy modeling [73-81]. Fuzzy modeling uses a natural descriptive language to form a system model based on fuzzy logic with fuzzy predicates.

The knowledge representation in fuzzy modeling can be viewed as having two classes. The first (class A), as suggested by *Takagi and Sugeno* in [19], can represent a general class of static or dynamic nonlinear systems. It is based on "fuzzy partition" of input space and it can be viewed as the expansion of a piecewise linear partition which is represented as

$$R^i: \textit{If } x_1 \textit{ is } A_1^i \textit{ and } x_2 \textit{ is } A_2^i, \ldots, \textit{and } x_m \textit{ is } A_m^i \quad \textit{then } y^i = a_o^i + a_1^i x_1 + \ldots + a_m^i x_m \tag{3.1}$$

where R^i ($i=1, 2, \ldots, c$) denotes the ith fuzzy rule, and x_j ($j= 1,2, \ldots, m$) is the input and y^i is the output of the fuzzy rule R^i. $A_1^i, A_2^i, \ldots, A_m^i$ ($i=1,2,\ldots,c$) are fuzzy membership functions which can be bell-shaped, trapezoidal, or triangular, etc., and usually they are not associated with linguistic terms. From (3.1), it is noted that *Takagi and Sugeno* approach approximates a nonlinear system with a combination of several linear systems by decomposing the whole input space into several partial fuzzy spaces and representing each output space with a linear equation. This type of knowledge representation does not allow the output variables to be described in linguistic terms which is one of the drawbacks of this approach. Another drawback is that the parameter identification of this model is carried out iteratively using a nonlinear optimization method [19, 79]. The implementation of this method is not an easy task [77, 80, 81], as the problem of determining the optimal membership parameters involves a nonlinear programming problem.

The second class of knowledge representation (class B) in fuzzy models was developed by *Mamdani* [82] and used by *Lin and Lee* [40] and *Sugeno and Yasukawa* [76]. The knowledge is presented in these models as

$$R^i: \textit{If } x_1 \textit{ is } A_1^i \textit{ and } x_2 \textit{ is } A_2^i, \ldots, \textit{and } x_m \textit{ is } A_m^i \quad \textit{then } y^i \textit{ is } B^i \tag{3.2}$$

where $A_1^i, A_2^i, \ldots, A_m^i, B^i$ ($i=1,2,\ldots,c$) are fuzzy membership functions which are bell-shaped, trapezoidal, or triangular, etc., and usually associated with linguistic terms. This approach has some advantages over the first approach. The consequent parts are presented by linguistic terms, which makes this model more intuitive and understandable and gives more insight into the model structure. Also, this modeling approach is easier to implement than the first approach [81]. This second form (class B) of knowledge representation will be adopted throughout this work as we are more concerned with the linguistic modeling approaches.

Many studies regarding finding the rules and tuning the membership function parameters of fuzzy models have been reported [73-81]. Neural networks are integrated with fuzzy logic in a form of Fuzzy Neural Networks (FNNs) and used to build fuzzy models [40, 83-88]. Many algorithms have been proposed to train these FNNs [83-88], and *Jang et al.* [85] have reviewed the fundamental and advanced developments in neuro-fuzzy synergisms for modeling and control. However, efficient

design of fuzzy models/controllers that provides both interactivity with humans as well as accuracy, still needs more considerable investigation and focusing research.

In this chapter, an outline of a systematic approach for building fuzzy models/controllers with an optimized (efficient) performance is proposed. The suggested model topology, that can support the incorporation of both qualitative and quantitative information during the identification process, is described in detail. The proposed approach integrates fuzzy logic, neural networks, and genetic algorithms in a systematic way to utilize their strengths.

3.2 The neuro-fuzzy model topology

In this thesis, the Neuro-Fuzzy (NF) model is built using the multilayer fuzzy neural network shown in Figure 3.1. The system has a total of five layers as proposed by *Lin and Lee* [40]. A model with two inputs and a single output is considered here for convenience. Accordingly, there are two nodes in layer 1 and one node in layer 5. Nodes in layer 1 are input nodes that directly transmit input signals to the next layer. Layer 5 is the output layer. Nodes in layers 2 and 4 are "term nodes" and they act as membership functions to express the input/output fuzzy linguistic variables. A bell-shaped function is adopted to represent a membership function, in which the mean value m and the variance σ are adjusted through the learning process. The two fuzzy sets of the first and the second input variables consist of n_1 and n_2 linguistic terms, respectively. The linguistic terms, such as positive large (PL), positive medium (PM), positive small (PS), zero (ZE), negative small (NS), negative medium (NM), negative large (NL), are numbered in descending order in the term nodes. Hence, n_1+n_2 nodes and n_3 nodes are included in layers 2 and 4, respectively, to indicate the input/output linguistic variables.

Each node of layer 3 is a "rule node" and represents a single fuzzy control rule. In total, there are $n_1 \times n_2$ nodes in layer 3 to form a fuzzy rule base for two linguistic input variables. The links of layers 3 and 4 define the preconditions and consequences of the rule nodes, respectively. For each rule node, there are two fixed links from the input term nodes. Layer 4 links, encircled in dotted line, are adjusted in response to varying control situations. By contrast, the links of layers 2 and 5 remain fixed between the input/output nodes and their corresponding term nodes. The NF model can adjust the fuzzy rules and their membership functions by modifying layer 4 links and the parameters that represent the bell-shaped membership functions for each node in layers 2 and 4. For convenience we use the following notation to describe the functions of the nodes in each of the five layers:

net_i^L : the net input value to the i-th node in layer L,

O_i^L : the output value of the i-th node in layer L,

m_i^L, σ_i^L : the mean and variance of the bell-shaped function of the i-th node in layer L,

W_{ij} : the link that connects the output of the j-th node in layer 3 with the input to the i-th node in layer 4.

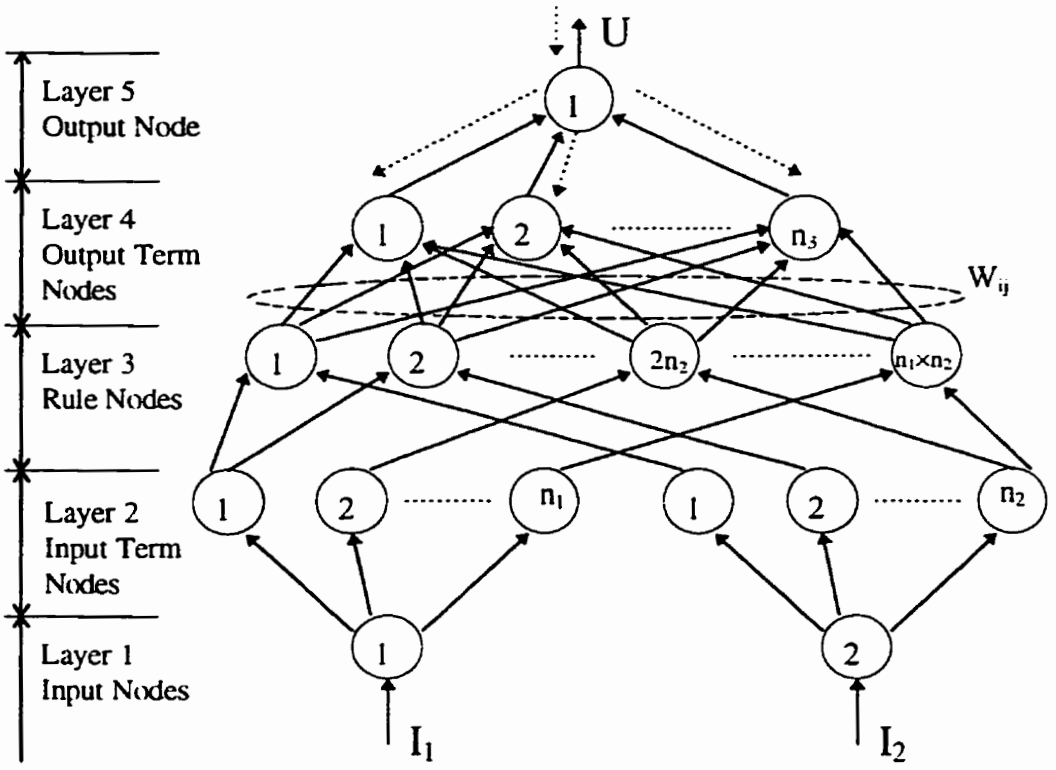


Figure 3.1 Topology of the neuro-fuzzy model.

Layer 1: The nodes of this layer directly transmit input signals to the next layer. That is,

$$O_1^1 = I_1 , \; O_2^1 = I_2 \tag{3.3}$$

Layer 2: The nodes of this layer act as membership functions to express the linguistic terms of input variables. For a bell-shaped membership function, they are

$$net_i^2 = \begin{cases} O_1^1 & for\ i = 1,2, \ldots\ldots, n_1 \\ O_2^2 & for\ i = n_1+1, \ldots\ldots, n_1+n_2 \end{cases} \tag{3.4}$$

$$O_i^2 = e^{-\left(\frac{net_i^2 - m_i^2}{\sigma_i^2}\right)^2} \qquad for\ i = 1,2,\ldots\ldots, n_1+n_2 \tag{3.5}$$

Note that layer 2 links are all set to unity.

Layer 3: The links in this layer are used to perform precondition matching of fuzzy rules. Thus, each node has two input values from layer 2. The correlation-minimum inference procedure is utilized here to determine the firing strengths of each rule. The output of the nodes in this layer is determined by fuzzy AND operation. Hence, the functions of the layer are as follows:

$$net_i^3 = \min(O_j^2, O_k^2),\ i = n_1(j-1)+(k-n_2) \qquad for\ j = 1,2, \ldots\ldots, n_1;\ k = n_1+1, n_1+2, \ldots\ldots, n_1+n_2 \tag{3.6}$$

$$O_i^3 = net_i^3 \qquad for\ i = 1,2,\ldots\ldots, n_1 * n_2 \tag{3.7}$$

The link weights in this layer are also set to unity.

Layer 4: Each node of this layer performs the fuzzy OR operation to integrate the field rules leading to the same output linguistic variable. The functions of the layer are expressed as follows:

$$net_i^4 = \sum_{j=1}^{n_1 * n_2} W_{ij} O_j^3 \tag{3.8}$$

$$O_i^4 = \min(1, net_i^4) \qquad for\ i = 1,2, \ldots, n_3 \tag{3.9}$$

The link weight W_{ij} in this layer expresses the association of the j-th rule with the i-th output linguistic variable. It can take only two values; either 1 or 0.

Layer 5: The node in this layer computes the output signal of the NF model. The output node together with layer 5 links act as a defuzzifier. The center of area defuzzification scheme [17-18], used in this model, can be simulated by

$$net_1^5 = \sum_{j=1}^{n_3} m_j^4 \sigma_j^4 O_j^4 \tag{3.10}$$

$$O_1^5 = \frac{net_1^5}{\sum_{j=1}^{n_3} \sigma_j^4 O_j^4} \tag{3.11}$$

Hence, the *j-th* link weight in this layer is $m_j^4 \sigma_j^4$.

3.3 The hybrid learning scheme

In this section, a three-phase learning scheme for the proposed neuro-fuzzy connectionist model is presented [87]. In phase one, unsupervised learning algorithms are used to locate the initial membership functions by clustering the data into an appropriate number of clusters. In phase two, supervised as well as unsupervised learning algorithms are used to find the fuzzy rules. In phase three, supervised learning algorithms are used to optimally adjust (fine-tune) the input/output membership functions. To initiate the learning scheme, training data must be provided from the outside world.

3.3.1 Learning Phase One

The problem for the first learning phase can be stated as: "Given the training input data $x_i(t)$, $i=1, ...,n$, $t=1, ..., N$ *(no. of training examples)*, the desired output value $y_i(t)$, $i=1,2, ...,m$, the fuzzy partitions |T(x)| and |T(y)|, and the desired shapes of membership functions, we want to locate the initial membership functions". In this phase, the network works in a two-sided manner; that is, the nodes and the links at layer four are in the up-down transmission mode (follow the dotted arrows in Figure 3.1) so that the training input and output data are fed into this network from both sides.

To form the membership functions, this learning phase clusters each input/output vector in the training data into an appropriate number of clusters, which could be determined from the designer's experience or using assessment techniques (clustering validity measures) for measuring clustering quality. The number of clusters represents the required fuzzy partition of the input/output space (i.e., the size of the term set of each input/output linguistic variable).

There are many clustering techniques that could be used in this learning phase.

However, the following candidates present some of the most suitable for building fuzzy membership functions:

1. Kohonen Self-Organizing feature Maps (SOM) [47].
2. Fuzzy C-Means clustering (FCM) [89].
3. Mountain function algorithm [90].
4. Subtractive clustering [91].

Mountain function as well as subtractive clustering algorithms are usually used to serve as tools to obtain the initial estimation of the cluster centers, which can be fine-tuned after that using Kohonen self-organizing feature maps or fuzzy c-means clustering algorithms. However, in situations where only approximate, not too exact, values of cluster centers are needed, these approaches can act as a stand alone clustering algorithm. Accordingly, the study in this work will focus on the first two algorithms as they give more accurate clustering, and these algorithms will be initiated by distributing the initial clusters centers regularly within the clustering space. The details of this study will be presented in Chapter 4.

3.3.2 Learning Phase Two

After the initial parameters of the membership functions have been found, the training signals from both external sides can reach the outputs of term nodes at layer two and layer four. Furthermore, the outputs of term nodes at layer two can be transmitted to rule-nodes through the initial architecture of layer-three links. Thus we can get the firing strength of each rule node. Based on these rule firing strengths (denoted as $O_{ki}^{3}(t)$'s) and the outputs of term nodes at layer four (denoted as $O_{ki}^{4}(t)$'s), we want to decide the correct consequence-link for each rule node (from the connected n_3 layer-four-links) to find the $n_1 \times n_2$ fuzzy rules, by one of the following learning algorithms:

1. Competitive Learning Algorithms (CLA) [92].
2. Minimum Distance Algorithm (MDA) [87].
3. Maximum Matching Factor Algorithm (MMFA) [93].
4. Static Genetic Algorithm (SGA) [94].

After applying one of the above learning algorithms using the whole training data set, the link weights at layer four represent the strength of the existence of the corresponding rule consequence. Among the links which connect a rule node (layer 3) and the term nodes (layer 4) of the output linguistic node, at most one link with maximum weight is chosen and the others are deleted. Hence, only one term in an output linguistic variable's term set can become one of the consequences of a fuzzy

rule. If all the link weights between a rule node and the term nodes of an output linguistic node are very small, then all the corresponding links are deleted, meaning that this rule node has little or no relation to this output linguistic variable. If all the links between a rule node and the layer-four nodes are deleted, then this rule node can be eliminated since it does not affect the outputs.

After the consequences of rule nodes are determined, the rule combination is performed to reduce the number of rules. The criteria for a set of rule nodes to be combined into a single rule node are: 1) they have exactly the same consequences, 2) some preconditions are common to all the rule nodes in this set, 3) the union of other preconditions of these rule nodes composes the whole term set of some input linguistic variables.

The details of these algorithms as well as a comprative study between them will be presented in Chapter 5.

3.3.3 Learning Phase Three

After the fuzzy rules are found, the whole network structure is established, and the third-learning phase is started in order to optimally adjust the parameters of the membership functions. Optimization, in the most general form, involves finding the most optimum solution from a family of reasonable solutions according to an optimization criterion. For all but a few trivial problems, finding the global optimum can never be guaranteed [95, ch. 14]. Hence, optimization in the last three decades has focused on methods to achieve the best solution per unit computational cost.

The problem for this supervised learning phase can be stated as: "Given the training input data $x_i(t)$, $i=1, ...,n$, the desired output value $y_i(t)$, $i=1,2, ...,m$, the fuzzy partitions |T(x)| and |T(y)|, the desired shapes of membership functions, the initial parameters of the membership functions, and the fuzzy rules, adjust the parameters of the membership functions optimally". In this phase, the network works in the feedforward manner; that is, the nodes and the links at layer four are in the down-up transmission mode (follow the solid lines in Figure 3.1). The proposed idea is to use one of the following algorithms:

1. Backpropagation algorithm [96].
2. Multi-Resolutional Dynamic Genetic Algorithm (MRD-GA) [97].

The details of these algorithms as well as a comparative study between them will be presented in Chapter 6. The whole three-phase learning scheme is summarized in the following flow chart (Figure 3.2).

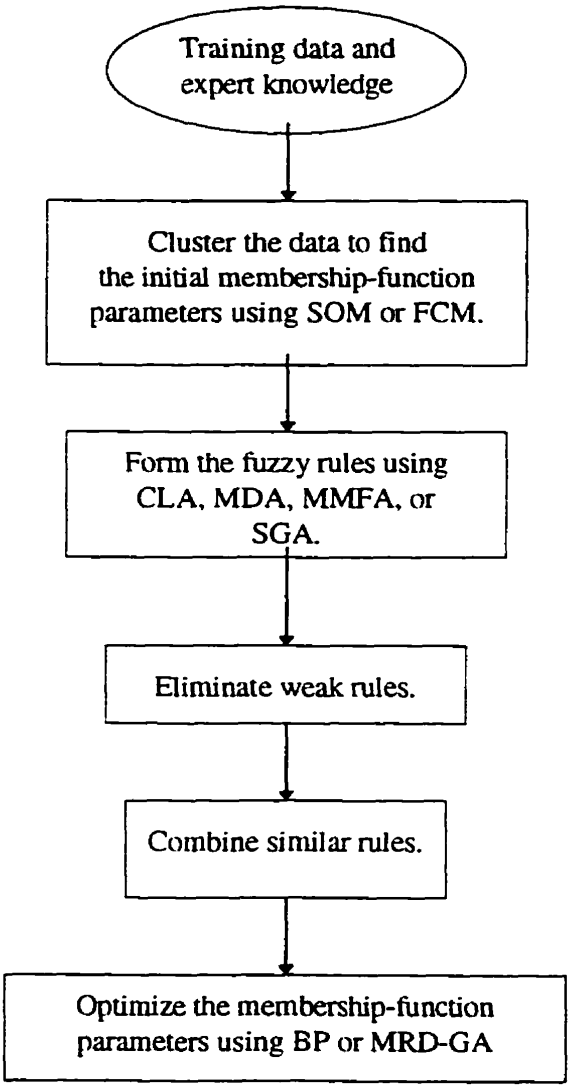


Figure 3.2 Flow of the hybrid learning scheme.

Chapter 4

Coarse Identification Phase

4.1 Introduction

By looking at the problems included in the identification of fuzzy models/controllers, we can divide the identification into two levels: structure-identification and parameter-identification. The structure identification of a system has to solve two problems: in the first one, one has to find the input variables and in the second one, one has to find the most appropriate fuzzy partitioning of each input/output variable.

In ordinary system identification, parameters are the coefficients in a functional system model. In a fuzzy model, the parameters are those in the membership functions of fuzzy sets. There is no big difference except in the number of parameters, that are much more in fuzzy identification. In principle, the structure identification problem and parameter identification problem can't be separately solved, which makes the identification task very complicated. In our approach, we simplify the identification by solving the parameter identification problem after the structure identification one.

Clustering is a tool that attempts to assess the relationships among patterns of a data set by organizing the patterns into groups or clusters such that patterns within a cluster are more similar to each other than are patterns belonging to different clusters. Clustering is used here to form the initial membership functions for the input/output variables. For each variable, each cluster represents a linguistic term and the number of clusters represents the fuzzy partition.

The unsupervised neural-network algorithms are available candidates to find clusters of data that can represent fuzzy membership functions. One of these candidates is Kohonen's Self-Organizing feature Maps (SOM) algorithm [47] that

constructs internal models to capture regularities in their input data vectors without receiving any additional information from the outside world.

Another available candidate is the Fuzzy C-Means (FCM) algorithm proposed by *Bezdek* [89] in 1981. He has suggested the use of an objective function approach for clustering the data into hyperspherical clusters. Both of the SOM and FCM algorithms are adapted in this chapter to solve the parameter identification problem by finding the initial centers *m's* and widths σ*'s* of the bell-shaped membership functions of fuzzy models.

Also, in this chapter, it is suggested to solve the structure identification problem using, what's called, Clustering Validity Measures (CVMs) to assess the quality of clustering for each input/output variable. Three CVMs are presented here with a comparison between them. Also, a detailed assessment of each clustering algorithm together with a comparative study are presented.

4.2 Self-Organizing Feature Maps

In the Kohonen's Self-organizing feature Map (SOM) that is shown in Figure 4.1, the four output neurons are arranged on a 2-dimensional lattice (higher-dimensional maps can generally be defined). Each input vector component is connected to each neuron via a synaptic weight that is calculated in the training mode. Each neuron possesses a weight vector that has the same size as the input vector.

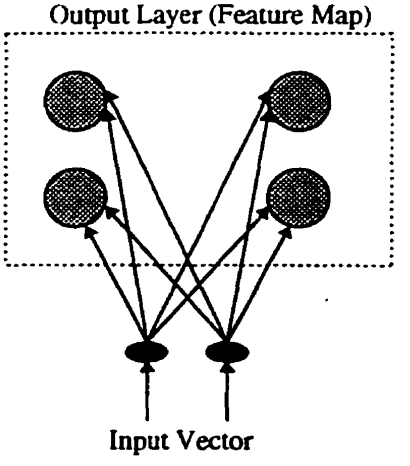


Figure 4.1 Neural network architecture of Kohonen's self-organizing feature map.

The purposes of the SOM algorithm are to cluster the input space into a finite number of classes represented by the neural-network weight-vectors, and to perform a topology-preserving mapping of high dimensional input vectors (i.e., long vectors) onto a lower dimensional surface represented by the location of the neuron on the grid.

In the SOM network, unsupervised learning is achieved in the feature map layer through competition. When an input pattern from the training set is presented to the network each neuron in the feature map layer computes the relative Euclidean distance between its weight vector and the input vector. The units then compete for the privilege of learning. Thus, the unit with the minimum Euclidean distance is chosen as the winning unit. This unit and its immediate neighbors on the grid are the only units permitted to learn in this pattern presentation by updating their weight vectors. During training, the weight adjustment is proportional to the difference between the input vector and weight vector.

After training, any input vector stimulates only the neuron whose weight vector is the closest to the input vector in the input space. The weight vectors, therefore, represent certain averages of disjoint set or classes of input vectors.

The problem of finding the membership-function parameters of a fuzzy model can be stated as: "Given the input vectors $x_i(t)$, $i=1, \ldots, N$, the desired fuzzy partitioning, and the desired shapes of membership functions, we want to initially locate the membership functions".

The bell-shaped function is considered here to represent fuzzy membership functions due to its simplicity, continuity and adaptability. This function is given by the following equation:

$$f_i(x) = e^{-\left(\frac{\|x-m_i\|}{\sigma_i}\right)^2} \tag{4.1}$$

Using the above function, the SOM algorithm is adapted to find the center m_i and the width σ_i of the i-th membership function by the following equations [40]:

$$\|x(t) - m_{closest}(t)\| = \min_{1 \le i \le c} \|x(t) - m_i(t)\| \tag{4.2}$$

$$m_{closest}(t+1) = m_{closest}(t) + \alpha(t)\left[x(t) - m_{closest}(t)\right] \tag{4.3}$$

$$m_i(t+1) = m_i(t) \quad for\ m_i \neq m_{closest} \tag{4.4}$$

where $\alpha(t)$ is a monotonically decreasing scalar learning rate, c is the required fuzzy partitioning, and t is the iteration counter.

Once the centers of membership functions are found, their widths can be determined by the N-nearest-neighbors heuristic, by minimizing the following objective function with respect to the widths σ_i's, i.e.,

$$E = \frac{1}{2}\sum_{i=1}^{N}\left[\sum_{j\in N_{neurs}}\left(\frac{m_i - m_j}{\sigma_i}\right)^2 - r\right]^2 \tag{4.5}$$

where r is an overlap parameter that usually ranges from 1.0 to 2.0. Since it is difficult to find the widths from Equation (4.5), they can be roughly determined by the first-nearest-neighbor heuristic as

$$\sigma_i = \frac{\left|m_i - m_{closest}\right|}{r} \tag{4.6}$$

4.3 Fuzzy C-Means Clustering

The Fuzzy C-Means (FCM) clustering [89] is the fuzzy equivalent of the nearest mean "hard" clustering algorithm [98], which minimizes the following objective function with respect to fuzzy membership μ_{ij} and cluster centroid m_i,

$$J_n = \sum_{i=1}^{c}\sum_{j=1}^{N}\left(\mu_{ij}\right)^n \left\|x_j - m_i\right\|^2 \tag{4.7}$$

where c is the number of clusters, N is the number of input vectors (data points), and $n > 1$ is the fuzziness index [89]. The FCM algorithm is performed in the following steps.

Step 1: Initialize memberships μ_{ij} of x_j belonging to cluster i such that

$$\sum_{i=1}^{c}\mu_{ij} = 1 \tag{4.8}$$

Step 2: Compute the fuzzy centroid m_i for $i = 1, 2, ..., c$ using

$$m_i = \frac{\sum_{j=1}^{N}\left(\mu_{ij}\right)^n x_j}{\sum_{j=1}^{N}\left(\mu_{ij}\right)^n} \tag{4.9}$$

Step 3: Update the fuzzy membership μ_{ij} using

$$\mu_{ij} = \frac{\left\| x_j - m_i \right\|^{\frac{-2}{(n-1)}}}{\sum_{i=1}^{c} \left\| x_j - m_i \right\|^{\frac{-2}{(n-1)}}} \tag{4.10}$$

Step 4: Repeat steps 2 and 3 until the value of J_n is no longer decreasing.

After the algorithm converges to strict local minima of J_n, the resultant m_i's are the optimal centers of the bell-shaped membership functions in the fuzzy model. In order to find the widths (σ_i's) of the membership functions, the following algorithm is used:

For i= 1 to c

Define the cost function $f_i(\sigma_i) = \sum_{j=1}^{N} \left(e^{-\left(\frac{\left| x_j - m_i \right|}{\sigma_i}\right)^2} - \mu_{ij} \right)^2$,

Find the σ_i *value that minimizes the cost function* $f_i(\sigma_i)$.
End

The σ_i that minimizes the cost function can be found using the well-known Least Mean Squares (LMS) algorithms.

4.4 Clustering Validity Measures

The subjective nature of the clustering problem precludes a realistic mathematical comparison of all clustering techniques. Hence, an intimately related important issue to the clustering problem is the "cluster validity" which deals with the significance of the structure imposed by a clustering method. A cluster validity function is used to measure the quality of a clustering by measuring how closely the data points are associated to the cluster centers. The level of association or classification can be measured by defining a membership function for each cluster. If the value of one of the memberships is significantly larger than the others for a particular data point, then that point is identified as being a part of the subset of the data presented by the corresponding cluster center.

As a fuzzy clustering validity function *Bezedek* [89] designed the partition coefficient *S1* to measure the amount of "overlap" between clusters.

$$S1 = \frac{1}{N}\sum_{i=1}^{c}\sum_{j=1}^{N}(\mu_{ij})^2 \tag{4.11}$$

In this form, *S1* is inversely proportional to the overall average overlap between pairs of fuzzy subsets. In particular, there is no membership sharing between any pairs of fuzzy clusters if $S1 = 1$. Finding the c value that produces the maximum *S1* value is the method to find a valid (appropriate) clustering of any data set. Disadvantages of the partition coefficient *S1* are the lack of direct connection to the geometrical property of the data and its monotonic-decreasing tendency with the increase of c.

Another fuzzy clustering validity function is proposed by *Xie et al* [99]. This function measures the overall average compactness and separation of a fuzzy c-partition. In this function, the separation of the fuzzy c-partition is defined with the parameter s, where

$$s = (d_{\min})^2 \tag{4.12}$$

$$d_{\min} = \min_{i,j}\|m_i - m_j\|. \tag{4.13}$$

A larger s indicates that all the clusters are more separated. The average compactness measure of the data in all the fuzzy subsets is given by

$$\pi = \frac{1}{N}\sum_{i=1}^{c}\sum_{j=1}^{N}(\mu_{ij})^2\|x_i - m_j\|^2 \tag{4.14}$$

The compactness and separation validity function *S2* is defined as the ratio of compactness π to the separation s. The overall expression is given by

$$S2 = \frac{\sum_{i=1}^{c}\sum_{j=1}^{N}\mu_{ij}^2\|m_i - x_j\|^2}{N * \min_{i,j}\|m_i - x_j\|^2} \tag{4.15}$$

As a special case, using the FCM algorithm with n=2, *S2* can be shown to be

$$S2 = \frac{J_2}{N * (d_{\min})^2} \tag{4.16}$$

which is very easy to calculate. From (4.15) and (4.16), it is clear that the smaller the validity measure *S2* the better is the fuzzy clustering, and the smallest *S2* indicates a valid optimal fuzzy partition.

The third validity criterion presented here is proposed by *Sugeno et al* [76]. They used a validity function defined by the following equation:

$$S3 = \sum_{j=1}^{N}\sum_{i=1}^{c}\left(\mu_{ij}\right)^{n}\left(\left\|x_j - m_i\right\|^2 - \left\|\bar{x} - m_i\right\|^2\right) \qquad (4.17)$$

where $\bar{x}$ is the average of the data (x_i, $i=1, 2, 3, \ldots\ldots, N$).

As seen in (4.17), the first term of the right-hand side is the variance of the data in a cluster and the second term is that of the clusters themselves. Therefore the optimal clustering is expected to minimize the variance in each cluster and to maximize the variance between clusters. So, the minimum value of *S3* indicates an optimal valid fuzzy clustering.

4.5 Testing and Evaluation Results

Three different well-known benchmarks are used in this section to compare between the SOM and FCM algorithms, and evaluate their effectiveness in building fuzzy membership functions. The benchmarks are also used to investigate the effectiveness of the cluster validity measures described above. In each benchmark, the number of clusters in the data is already known in advance. Also, it is known to which cluster each point in the data set belongs. Thus, the results found by each algorithm are compared with the actual results.

4.5.1 Example 1

The well-known Anderson's Iris data [100] consists of 150 four-dimensional vectors. The components of a vector are the measurements of the petal length, petal width, sepal length, and sepal width of a particular Iris plant. There are 50 flowers in each of the three subspecies of Iris represented in the data, so it is assumed that an effective validity function should indicate the presence of three clusters. In the numerical representation of the data, two of the classes have substantial overlap, while the third is well separated from the other two (linearly separable). Thus, one can argue in favor of both $c=2$, and $c=3$ for this data as a valid clustering. We have divided this data into two equal groups. The first is used for training, and the other is used for testing.

The SOM algorithm (presented in Section 4.2) is applied to cluster the Iris training data with $r = 1.5$. The three validity measures (*S1*, *S2*, and *S3*) are used to identify the number of clusters in the data. The results of the analysis are shown in Table 4.1. The

maximum of *S1* is produced at $c^* = 3$, which matches the actual number of clusters. However, the minimum values of both *S2* and *S3* are produced at $c^*=2$ and $c^*=5$, respectively. Both of them failed to match the actual clustering. However, *S2* produced more accepted clustering partition as mentioned before. The learning and the testing results of the Iris data using the SOM are summarized in Table 4.2, which shows a performance of 88%. The results show that the algorithm performance depends on the resultant centers $m_i s$ and not on the value of the parameter r (which is used to evaluate the widths $\sigma_i s$ using equation (4.6)).

Table 4.1 Validity measures of Iris data after SOM (r =1.5).

No. of Clusters	*S1*	*S2*	*S3*
2	0.0674	0.0207	0.0245
3	0.2692	0.0833	-1.5772
4	0.1798	0.1573	-3.0247
5	0.2016	0.0729	-4.1034
6	0.1531	0.0735	-3.3706
7	0.0876	0.0702	-1.4447

Table 4.2 SOM results of Iris Data.

r	75 Learning Samples		75 Testing Samples	
	Hit	Miss	Hit	Miss
1.0	65	10	67	8
1.5	65	10	67	8

Then, the FCM algorithm (presented in Section 4.3) is applied to cluster the Iris data with $n = 2$, $\varepsilon = 10^{-6}$, and $c = 2, 3 \ldots 7$. The three validity measures (*S1*, *S2*, and *S3*) are used to identify the number of clusters in the data. The results of the analysis are shown in Table 4.3. The minimum of *S3* is produced at $c^* = 3$, which matches the actual number of clusters. However, both the minimum value of *S2* and the maximum value of *S1* are produced at $c^*=2$, which are also acceptable results. The learning and the testing results of the Iris data using the FCM algorithm are summarized in Table 4.4, which shows a performance of 90%. The resultant widths $\sigma_i s$ do not affect the results on Table 4.4. However, they affect the validity measures values. These widths are found as proposed in Section 4.3. Table 4.5 shows a rough comparison between the speed of both the SOM and FCM algorithms in the clustering of the Iris data at $c = 3$. This comparison is just presented as an indication of the relative speed between the two algorithms.

Table 4.3 Validity measures of Iris data after FCM.

No. of Clusters	*S1*	*S2*	*S3*
2	0.8653	0.0723	-8.9163
3	0.7482	0.1744	-11.4572
4	0.6638	0.2841	-11.0526
5	0.6027	0.5290	-10.3867
6	0.5646	0.3555	-10.2185
7	0.5246	0.3535	-9.9166

Table 4.4 FCM results of Iris data.

75 Learning Samples		75 Testing Samples	
Hit	Miss	Hit	Miss
68	7	67	8

Table 4.5 The speed of both SOM and FCM.

	SOM	FCM
No. of Flops	1,505,352	128,039
No. Iterations	10,000	15
Flops/Iteration	151	8536

4.5.2 Example 2

The second example uses an artificially generated data proposed by *Windham* [101]. The data set consists of a two-dimensional data pictured in Figure 4.2. It is obtained by choosing 50 points at random in each of the disks of radius one centered at the points (2, 1), (2, -1), (-2, 1), and (-2, -1), respectively. As the figure indicates, it would be reasonable to expect that a clustering algorithm would identify the presence of four clusters. However, the presence of only two clusters is justifiable. A set of 200 different data points is also generated in the same way for the testing purposes.

The SOM algorithm is used with $r = 1.0$, and 2.0. The results of the analysis are shown in Table 4.6. *S1*, *S2*, and *S3* have failed to produce an optimal number of clusters that matches the actual number of clusters ($c = 4$), but *S2* has a justifiable result. The clustering and testing results are summarized in Table 4.7. The results show that the SOM performance is around 67%.

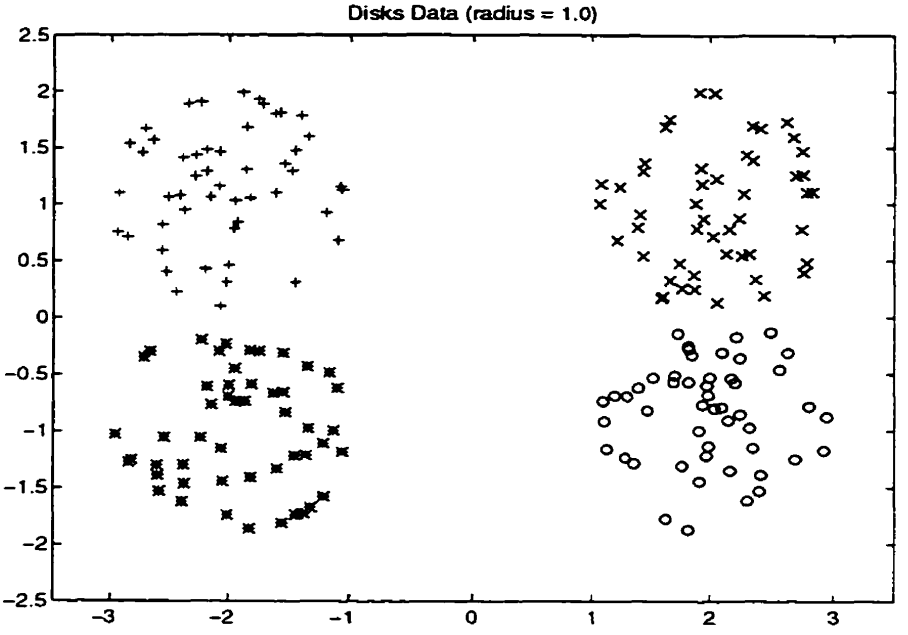


Figure 4.2 Disks data distribution (radius=1.0) of example 2.

Table 4.6 Validity measures of disks data (radius=1.0) after SOM.

No. of Clusters	*S1*		*S2*		*S3*	
	r = 1	*r = 2*	*r = 1*	*r = 2*	*r =1*	*r = 2*
2	0.0908	0.0013	0.0781	0.0013	19.4651	0.1842
3	0.6519	0.2087	0.2370	0.0229	-243.07	-145.00
4	0.3841	0.1100	0.3437	0.0268	-210.56	-76.477
5	0.3474	0.0835	1.0006	0.0285	-133.28	-60.714
6	0.5438	0.1347	0.9974	0.0528	-336.71	-116.45
7	0.6860	0.2011	0.7004	0.0379	-484.51	-180.56

Table 4.7 SOM results of disks data (radius=1.0).

r	200 Learning Samples		200 Testing Samples	
	Hit	Miss	Hit	Miss
2.0	135	65	132	68

Table 4.8 Validity measures of disks data after FCM.

No. of Clusters	*S1*	*S2*	*S3*
2	0.8627	0.0839	-409.14
3	0.7953	0.1512	-569.74
4	0.7559	0.1162	-679.22
5	0.6841	0.3142	-628.54
6	0.6437	0.3464	-600.25
7	0.5895	0.3813	-570.78

Thus, the FCM algorithm is used to cluster the data with $n = 2$, $\varepsilon = 10^{-6}$, and $c = 2, 3, \ldots 7$, and the validity measures are used to identify the number of clusters. The results of the analysis are shown in Table 4.8. The minimum of *S3* is produced at $c^* = 4$, which matches the actual number of clusters. However, both *S1* and *S2* produced justifiable results but failed to match the actual clustering. The learning and the testing results of FCM algorithm are summarized in Table 4.9, showing a 99.5% performance. Table 4.10 shows a rough comparison between the speed of both SOM and FCM algorithms at $c = 4$.

Table 4.9 FCM results of disks data.

200 Learning Samples		200 Testing Samples	
Hit	Miss	Hit	Miss
199	1	199	1

Table 4.10 The speed of both SOM and FCM.

	SOM	FCM
No. of Flops	1,075,600	419,905
No. Iterations	10,000	22
Flops/Iteration	108	19087

4.5.3 Example 3

The data in this example is artificially generated exactly in the same way like the previous example, except that the radii of the disks are 1.5 rather than 1. As can be seen in Figure 4.3, because of the overlap of the disks, this data set appears to have two clusters rather than four. Thus, an optimal clustering of two is an acceptable result. Different 200-points data set is also generated in the same way for testing

purposes.

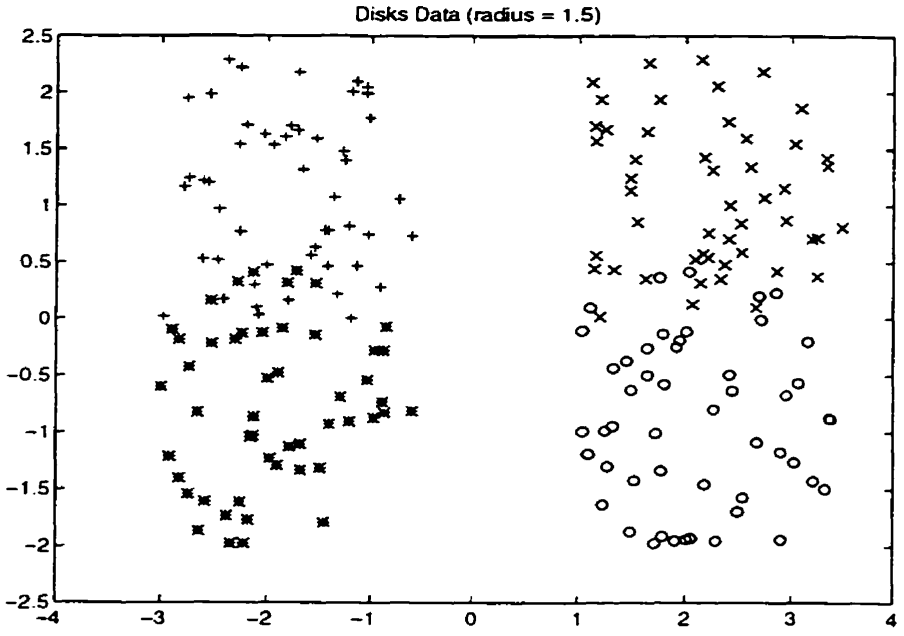


Figure 4.3 Disks data distribution (radius=1.5) of example 3.

The SOM algorithm is used with $r = 1.0$. The three validity measures are used to identify the number of clusters in the data. The results of the analysis are shown in Table 4.11. *S1*, and *S3* have failed to produce an optimal number of clusters that matches the actual number of clusters ($c = 4$), but *S2* produces an acceptable result. The clustering and the testing results are summarized in Table 4.12. The results show that the SOM performance is around 75%.

Table 4.11 Validity measures of disks data (radius = 1.5) after SOM.

No. of Clusters	*S1*	*S2*	*S3*
2	0.1343	0.0772	15.935
3	0.6033	0.2234	-203.82
4	0.3762	0.2618	-204.64
5	0.3246	0.7812	-117.05
6	0.4067	0.5219	-245.65
7	0.5334	0.6440	-289.49

Table 4.12 SOM results of disks data (radius = 1.5).

r	200 Learning Samples		200 Testing Samples	
	Hit	Miss	Hit	Miss
1.0	151	49	148	52

Then, the FCM algorithm is applied with $n = 2$, and $\varepsilon = 10^{-6}$, and the validity measures are used to identify the number of clusters. The results of the analysis are shown in Table 4.13. The minimum of *S3* is produced at $c^* = 4$, which matches the actual number of clusters. Both *S1* and *S2* produce acceptable results ($c = 2$). The results of FCM algorithm are summarized in Table 4.14, showing a 90% performance.

Table 4.13 Validity measures of disks data (radius=1.5) after FCM.

No. of Clusters	*S1*	*S2*	*S3*
2	0.8425	0.0996	-350.52
3	0.7086	0.2628	-425.29
4	0.6559	0.1494	-549.32
5	0.5990	0.2805	-525.40
6	0.5718	0.2077	-520.62
7	0.5487	0.1963	-536.94

Table 4.14 FCM results of disks data.

200 Learning Samples		200 Testing Samples	
Hit	Miss	Hit	Miss
183	17	178	22

5.4.4 The effect of *n*

In this section, we study the effect of n (the weighting exponent of the FCM algorithm) on the validity measures (*S1*, *S2*, and *S3*). The Iris data (150 samples) is used in this study. The data is clustered with different values of n ($n = 1 \ldots 7$) and different values of c ($c = 2 \ldots 10$). Table 4.15 lists the optimal values of c chosen by each of the validity measures. We have shaded those cells of the table which agree with the preferred (actual or acceptable) value of c for the Iris data.

Table 4.15 Optimal values of c chosen by each validity measure.

n	*S1*	*S2*	*S3*
1.1	2	2	9
1.5	2	2	10
2.0	2	2	3
2.5	2	2	3
3.0	2	2	3
4.0	2	6	3
5.0	2	6	10
6.0	2	4	9
7.0	2	4	10

From the analysis in Table 4.15, it is noticed that the most workable range of n is from 2.0 to 4.0, where n=2 has often been the preferred choice for many users [102]. The three validity measures have reasonable results within this range. The *S3* measure is extremely sensitive to the values of n outside this range. However, it produces the best results among the other measures within this range.

4.6 Conclusions

The SOM and FCM algorithms have been used for building fuzzy membership functions for three benchmarks. Table 4.16 shows a comparison between the two algorithms extracted from the results in this chapter. Both algorithms have good convergence and stability features. However, the performance of the FCM algorithm exceeds that of the SOM algorithm in all the evaluation tests.

Table 4.16 A comparison between SOM and FCM.

View point	SOM	FCM
Convergence & Stability	Good	Good
Performance: Example 1	88%	90%
Performance: Example 2	67%	99.5%
Performance: Example 3	75%	90%
Average Performance	77%	93%
Speed	Slower	Faster
Clustering assessment by *S1*	Inconsistent	Acceptable
Clustering assessment by *S2*	Inconsistent	Acceptable
Clustering assessment by *S3*	Inconsistent	Excellent

The speed of the SOM algorithm depends on the initial learning rate and the

required accuracy (ε). It is found that if the required accuracy of both the SOM and FCM algorithms are the same, the FCM algorithm is still faster. However, for small data sets (few hundreds of data points), the speed difference between the two algorithms is not of major concern, especially in off-line applications. The three validity measures (*S1*, *S2*, and *S3*) are used to assess the quality of clustering produced by each algorithm. The three measures failed to produce consistent results from the clusters found by the SOM algorithm. On the other hand, the *S1* and *S2* measures have produced acceptable results from those found by the FCM algorithm and *S3* measure has produced excellent results. Therefore, it can be concluded that the quality of clustering produced by the FCM algorithm is more appropriate than that of the SOM algorithm in building fuzzy membership functions, however, SOM algorithm is still useable.

The validity measure *S3* [76] shows an excellent performance with the FCM algorithm. The optimal fuzzy partition found by this measure always matches the actual partition in all the benchmarks used. However, this measure shows high sensitivity to the weighting exponent n. The suggested workable range of n is from 2.0 to 4.0. Beyond this range, *S3* has produced inconsistent results. This drawback of this measure is not of major concern if we know that $n=2.0$ is the most popular value used by many designers [102].

Finally, the FCM algorithm is recommended to be used in building fuzzy membership functions, and *S3* validity measure to be used in finding the most appropriate fuzzy partition (the optimal number of membership functions).

Chapter 5

Rule-Formation Phase

5.1 Introduction

In many of the real-world control problems, the human operator interaction is an essential part of the control loop. The environment facing this human controller, in such complex control systems, is so complicated that no mathematical model exists, or, the mathematical model is strongly nonlinear so that a systematic design method does not exist.

To design a controller that can replace the human operator in a control loop, we first need to see what information is available. Since in this work, we consider only a model-free design, there are usually two kinds of information available to us:

- The experience of the human controller that is usually expressed as some linguistic "IF-THEN" rules. These rules state in what situations which actions should be taken.
- Sampled input-output data pairs that are recorded from successful control by the human controller, or that give enough information about the dynamics of the plant.

The human experience alone is usually considered incomplete as, in many cases, some information will be lost when humans express their experience. Also, it is difficult for human experts to examine all input-output data recorded from a complex process to find, and tune the IF-THEN rules and their membership functions within fuzzy control systems [12]. Accordingly, there is a need for systematic approach to find the linguistic fuzzy rules from the input-output data of a complex plant. The required approach should decrease or eliminate the dependency on the human experience in forming the rules, but at the same time, should allow the incorporation of

this experience, in order to utilize all the available information in generating the rules.

In this chapter, four different techniques are presented to form the fuzzy-linguistic rules from the input/output data of a complex plant. These techniques are: the Competitive Learning Algorithm (CLA), Minimum Distance Algorithm (MDA), Maximum Matching-Factor Algorithm (MMFA), and Static Genetic Algorithm (SGA). A well-known benchmark is used to test the four techniques by building four different models for this benchmark. The performance of the four models is illustrated and, accordingly, each technique is evaluated. A comparison among the four techniques is also presented. Conclusions are drawn from the assessment of each technique.

5.2 Learning techniques

While fuzzy logic provides a mathematical morphology to emulate certain perceptual and linguistic attributes associated with human cognition, artificial neural networks offer exciting advantages such as learning, adaptation, fault-tolerance, and generalization. The similar parallelism properties of neural-nets and fuzzy-logic systems make their integration more suitable for solving and studying the behavior of imprecisely-defined complex systems [103]. The strengths of both systems can be utilized to form the fuzzy rules and tune their membership functions within a fuzzy controller [83-88].

In this section we present four learning schemes for finding the linguistic-fuzzy control rules. To initiate the learning schemes, training data and the desired or selected coarse of fuzzy partition must be provided from the outside world. Also, the initial centers (m's) and widths (σ's) of the membership functions have to be found. Chapter 4 has presented different techniques to find the initial membership functions.

After the parameters of the membership functions have been found, the training signals from both external sides can reach the outputs of the term nodes at layer 2 and layer 4. Furthermore, the outputs of term nodes at layer two can be transmitted to rule-nodes through the initial architecture of layer-three links. Thus, we can get the firing strength of each rule-node. Based on these rule-firing strengths (denoted as O_{ki}^{3}'s) and the outputs of term-nodes at layer four (denoted as O_{ki}^{4}'s), we want to decide the correct consequence-link for each rule node (from the connected n_3 layer-four-links) to find the $n_1 \times n_2$ rules. Four algorithms are presented here to perform this task.

5.2.1 CLA Algorithm [92]

Step 1: the links at layer four are initially fully connected. We denote the weight of the link between the *i*th rule node and *j*th output term node as w_{ij}, and initialize it with zero value. In this case, we have $n_1 \times n_2 \times n_3$ weights.

Step 2: w_{ij} is calculated according to the following pseudocode:

For $i = 1, 2, \ldots, n_1 \times n_2$
 For $j = 1, 2, \ldots, n_3$
  For $k = 1, 2, \ldots, N$ *(the no. of available training examples)*

$$w_{ij} = w_{ij} + O_{kj}^{4}(-w_{ij} + O_{ki}^{3})$$

  End
 End
End

Here O_{kj}^{4} serves as a win-loss index of the *j*th term node at layer 4. The theme of this law is that 'learn if win'. In the extreme case, if O_{kj}^{4} is '0-1' threshold function, then the above law says 'learn only if win'.

Step 3: After the competitive learning is performed through the whole training data set, the link-weights at layer 4 represent the strength of the existence of the corresponding rule consequence. Among the links, which connect rule node and term nodes of an output linguistic variable, at most one link with maximum weight is chosen and the others are deleted. Hence, only one term in an output linguistic variable's term set can become one of the consequences of a fuzzy logic rule. If all the link-weights between a rule node and the term nodes of an output linguistic node are very small, then all the corresponding links are deleted, meaning that this rule node has little or no relation to this output linguistic variable. If all the links between a rule node and the layer-four nodes are deleted, then this rule node can be eliminated since it does not affect the outputs.

5.2.2 MDA Algorithmm [87]

Step 1: We denote the Euclidean distance between the *i*th rule node and the *j*th output term node as d_{ij}, and initialize it with zero value. In this case, we have $n_1 \times n_2 \times n_3$ *d's*.

Step 2: d_{ij} is calculated according to the following pseudocode:

For $i=1, 2, ..., n_1 \times n_2$
 For $j = 1, 2, ..., n_3$
  For $k = 1, 2, ..., N$ *(the no. of available training examples)*
  if $O_{kj}^{3} > \eta_c$ *then* $d_{ij} = \sqrt{d_{ij}^2 + \left(O_{kj}^4 - O_{ki}^3\right)^2}$

  End
 End
End

where η_c is a specified threshold value. This value depends on the distribution of the training patterns, and is selected by heuristics for the best performance measure, as will be seen later in Section 5.3.

Step 3: After calculating all the d's using the previous code considering all the available training patterns, the rule-consequences can be determined form these factors according to the following pseudocode:

For $i = 1, 2, ..., n_1 \times n_2$
 Find the minimum distance d_{min} *from the set* d_i *([* d_{ij}*;* $j=1, 2, ..., n_3$*]),*
 Find the corresponding term-node index (j_{min}) *of* d_{min}*,*
 Delete all the layer-four-links of the i-th rule-node except the one connecting it with the term-node of index j_{min}*,*
End

5.2.3 MMFA Algorithm [93]

Step 1: For each layer-three-rule node we construct n_3 matching factors. In this case, we have $n_1 \times n_2 \times n_3$ matching factors. Each matching factor is denoted as M_{ij} with zero initial value, where the subscript i is the rule-node-index (i=1, 2, ..., $n_1 \times n_2$), and the subscript j is the output-linguistic-variable-index (output-term-node-index) (j=1, 2, ..., n_3).

Step 2: M_{ij} is calculated according to the following pseudocode:

For $i=1, 2, ..., n_1 \times n_2$
 For $j = 1, 2, ..., n_3$

For k = 1, 2, ... N (the no. of available training examples)

$$M_{ij} = \begin{cases} M_{ij} + O^3_{ki} & \text{if } O^4_{kj} \text{ is the maximum element in the set } O^4_k \\ M_{ij} & \text{Otherwise} \end{cases}$$

End

End

End

Step 3: After calculating all the M's using the previous code considering all the available training patterns, the rules consequences can be determined form these factors according to the following pseudocode:

For i = 1, 2, ..., $n_1 \times n_2$

Find the maximum matching-factor M_{max} *from the set* M_i *([* M_{ij}*; j=1, 2, ...,* n_3*]),*

Find the corresponding term-node index (j_{max}*) of* M_{max}

Delete all the layer-four-links of the i-th rule-node except the one connecting it with the term-node of index j_{max}

End.

Step 4: From the above algorithm, only one term in the output linguistic variable's term set can become the consequence of a certain fuzzy-logic rule. If all the matching factors of a rule-node are very small (meaning that this rule has small or no effect on the output), then all the corresponding links are deleted, and this rule-node is eliminated.

5.2.4 Static Genetic Algorithm [94]

The proposed SGA uses decimal-integer strings to encode the fuzzy rules. The decimal strings are considered a more suitable representative method than the binary strings. This representation allows the use of more compact-size strings. The number of alleles (individual locations which make up the string) is determined from the total number of fuzzy rules. From the NF model configuration shown in Figure 3.1, we have $n_1 \times n_2$ rules. It is allowed for each allele to take any value in the set [1, 2, ..., 9], where 1 represents NL, 2 represents NM, 3 represents NS, 4 represents NVS, 5 represents ZE, 6 represents PVS, 7 represents PS, 8 represents PM, and 9 represents PL.

The GA proposed here is called static to distinguish it from the dynamic one that will be proposed later in Section 6.3. The SGA is coded using the well-structured

language C++. The program allows the user to define the values for population size (pop_size), maximum number of generations (max_gen), probability of crossover (pcross), and probability of mutation (pmut). In order to select the individuals for the next generation, the tournament selection method is used. In this method, two members of the population are selected at random and their fitness values compared. The member with higher fitness advances to the next generation. An advantage of this method is that it needs less computational effort than other methods. Also, it does not need a scaling process (like the roulette wheel selection) [31]. However, the particulars of the reproduction scheme are not critical to the performance of the SGA; virtually, any reproduction scheme, in such kind of genetic algorithm, that biases the population toward the fitter strings works well [53].

The proposed SGA uses the Mean Squared Error (MSE) (the error is the difference between the actual output and the estimated output by the fuzzy model) as a fitness function. Simply, for each chromosome, (1/MSE) is considered as the fitness measure of it. The MSE is calculated from N data points as

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (u_i - \hat{u}_i)^2 \qquad (5.1)$$

where u_i is the actual value and $\hat{u}_i$ is the estimated value.

The usual GA terminating condition is a maximum number of allowable generations or a certain value of MSE required to be reached. In this SGA algorithm, the stopping criterion is the execution of a certain number of generations without any improvement in the best fitness value. In this criterion, you do not need to specify a required MSE value (which usually unknown in advance) or a required number of generations (where there is no guarantee that this number will produce an appropriate solution). This SGA uses simple crossover (single point crossover) and mutation operators.

The proposed SGA pseudocode is as follows.

Initialize P(t)→P(0). *: P(t) Population at time t.*
Initialize best_fit = 0. *: The best fitness value.*
Evaluate P(0).
Search for the best fitness of P(0) and assign best_fit to it.
While (not terminate-condition) do
    Begin
t← t+1 *:Increment generation.*

Select P(t) from P(t-1) using tournament selection criteria.
Recombine P(t) :apply genetic operators (crossover, mutation).
Evaluate P(t).
Search for the best fitness of P(t) and compare it with best_fit, if larger then do
Begin
Assign best_fit to the best fitness value of P(t).
End
End
End.

5.3 Simulation studies

The presented algorithms are examined using the well-known example of system identification given by *Box and Jenkins* [104]. The process is a gas furnace with a single input $u(t)$ and a single output $y(t)$: gas flow rate and CO_2 concentration, respectively. The data set consists of N=296 pairs of experimental input-output measurements. The sampling interval is 9 seconds.

The inputs to the fuzzy model are selected to be $y(t-1)$ and $u(t-4)$, respectively, as in [73]. The $y(t-1)$ input variable is modified to be $y'(t-1)$, where $y'(t-1) = y(t-1) - y_{mean}$, y_{mean} is the average of all the y's. Each of the input variables is partitioned into seven linguistic sets ($n_1=n_2=7$). The output of the fuzzy model is $y'(t)$, where the actual process output is $y(t) = y'(t) + y_{mean}$. The model output $y'(t)$ is partitioned into nine linguistic sets ($n_3=9$).

The gas furnace is modeled using the fuzzy-neural network shown in Figure 3.1. The SOM algorithm (Section 4.2) is used to determine the initial centers and widths of the 23 member functions of the input/output variables. The three scaling factors of this model are determined as 'Gu=0.658', 'Gy=0.227', and 'Go=7.909'. The resultant membership functions after finishing this learning phase are shown in Figure 5.1.

The CLA algorithm (Section 5.2.1) is then applied to find the fuzzy rules. Twenty-eight rules are only considered and shown in Table 5.1. The other 21 rules are deleted because their weights are very small (less than 5% of the highest weight). Usually, the rules with very small weights produce incorrect consequences due to the lack of adequate training examples in the region of these rules. Including these rules in the model may have detrimental effect. For example, including all the 49 rules that are found by the CLA algorithm produces an MSE of 2.3271. However, deleting the 21 weak rules results in an MSE of 2.1717. The blank rules could be filled by smoothing

out and/or extrapolating the existing rules. Leaving this rules blank implies a 'no action' state for the output and this sometimes gives better performance than filling the blank spaces with imprecise consequences. The output of the model with 28 rules is shown in Figure 5.2.

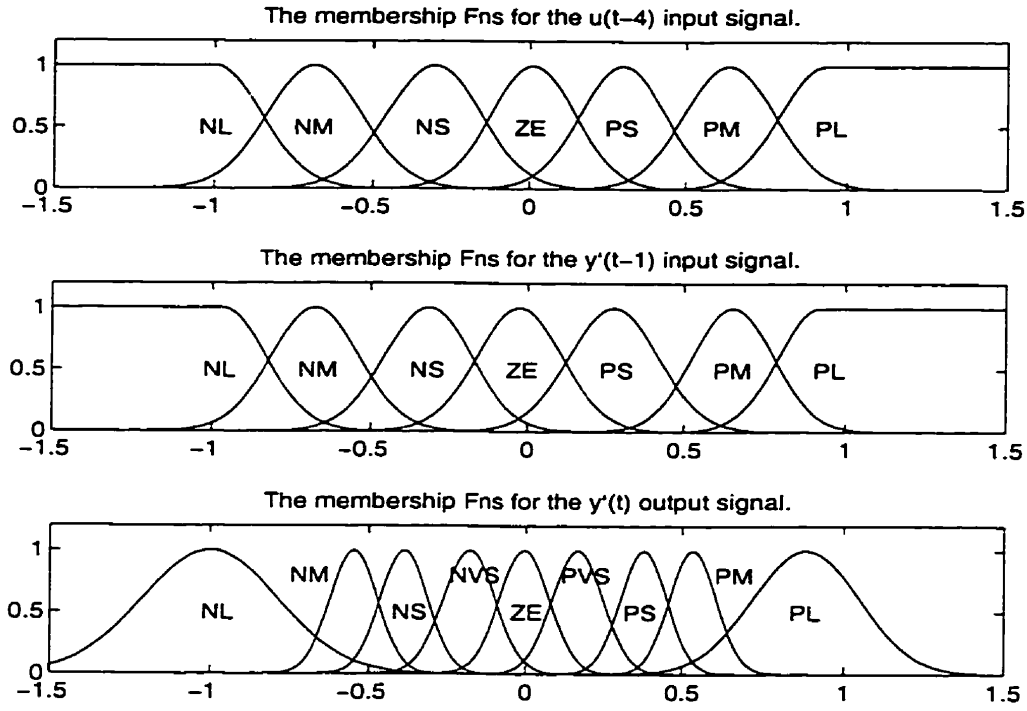


Figure 5.1 The normalized membership functions after SOM.

Table 5.1 The complete fuzzy associative memory matrix with CLA rules.

	u(k-4)						
y'(t-1)	NL	NM	NS	ZE	PS	PM	PL
NL					*NM*	*NM*	*NL*
NM			*NS*	*NS*	*NS*	*NM*	*NM*
NS			*ZE*	*NVS*	*NVS*	*NS*	
ZE		*PVS*	*PVS*	*NVS*	*NVS*		
PS		*PVS*	*PVS*	*PVS*			
PM		*PS*	*PS*	*PS*	*PS*		
PL	*PL*	*PL*	*PL*	*PM*	*PM*		

All entries correspond to y'(t)

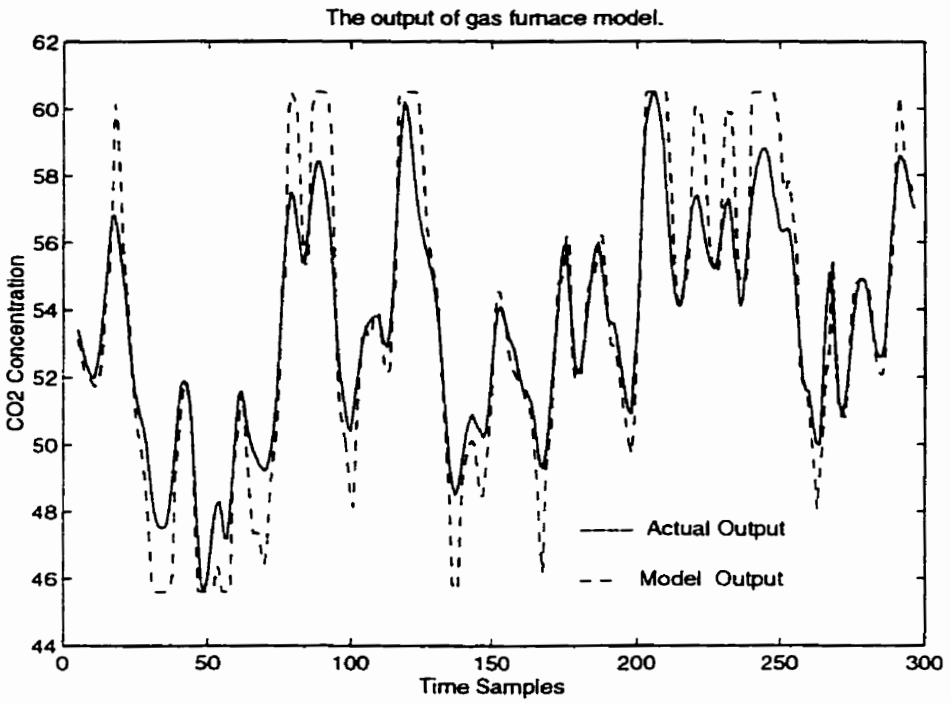


Figure 5.2 Output of the gas furnace model with CLA rules.

The MDA algorithm (Section 5.2.2) is then applied to find the fuzzy rules. Thirty-four rules are found and shown in Table 5.2. The other 15 rules are deleted as they are found to have insignificant effect. After many trials and simulation studies, it is found that the best MSE value of 0.9709 is produced at $\eta_c = 0.17$. The gas furnace output with MSE of 0.9709 is shown in Figure 5.3. The main disadvantage of this algorithm is the difficulty of finding the η_c value that gives the most appropriate set of rules. Table 5.3 shows the MSE values at different η_c's.

The purpose of using a threshold value η_c is to ensure that the training patterns, that are employed to find a certain rule, are in an active status. In other words, the algorithm uses only the training patterns that activate the rule node in layer 3, which is associated with a certain fuzzy rule. Employing all the training patterns (passive and active) deteriorate the algorithm performance as shown in Table 5.3 at $\eta_c = 0.0$.

Table 5.2 The complete fuzzy associative memory matrix with MDA rules.

	u(k-4)						
y(t-1)	NL	NM	NS	ZE	PS	PM	PL
NL					*NM*	*NM*	*NL*
NM			*NVS*	*NVS*	*NVS*	*NS*	*NM*
NS		*PVS*	*ZE*	*NVS*	*NVS*	*NVS*	*NVS*
ZE		*PVS*	*PVS*	*ZE*	*NVS*	*NS*	
PS	*PS*	*PS*	*PVS*	*PVS*	*PVS*	*ZE*	
PM	*PS*	*PS*	*PVS*	*PVS*			
PL	*PL*	*PM*	*PM*	*PM*			

All entries correspond to y(t)

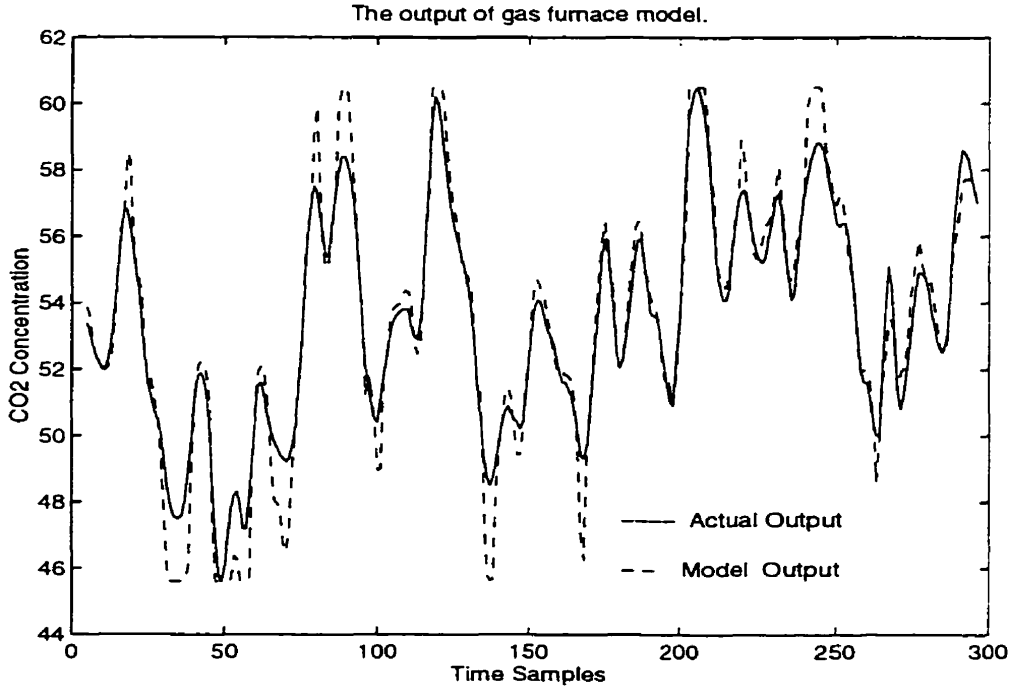


Figure 5.3 Output of the gas furnace model with MDA rules.

Table 5.3 The change of MSE with η_c.

η_c	0.0	0.05	0.07	0.11	0.15	0.17	0.2	0.25	0.35
MSE	41.05	4.0590	2.7514	1.4027	1.0173	0.9709	0.9740	0.9913	0.9962

The MMFA algorithm (Section 5.2.3) is used to find the linguistic fuzzy rules of the gas furnace model. Out of the 49 rules of the fuzzy model, 37 rules are only considered. The rest are deleted because they have very small matching factors (less than 5% of the highest matching factor of all the rules). The 37 rules are shown in Table 5.4. The model is simulated as shown in Figure 5.4, and its MSE value is found to be 0.9441.

Table 5.4 The complete fuzzy associative memory matrix with MMFA rules.

	u(k-4)						
$\dot{y}$(t-1)	NL	NM	NS	ZE	PS	PM	PL
NL				*NS*	*NS*	*NM*	*NL*
NM			*NVS*	*NVS*	*NVS*	*NS*	*NS*
NS		*ZE*	*ZE*	*NVS*	*NVS*	*NVS*	*NS*
ZE	*PVS*	*PVS*	*ZE*	*ZE*	*NVS*	*NVS*	
PS	*PS*	*PVS*	*PVS*	*ZE*	*ZE*	*ZE*	
PM	*PS*	*PS*	*PS*	*PVS*	*PVS*		
PL	*PL*	*PM*	*PS*	*PM*	*PM*		

All entries correspond to $\dot{y}$(t)

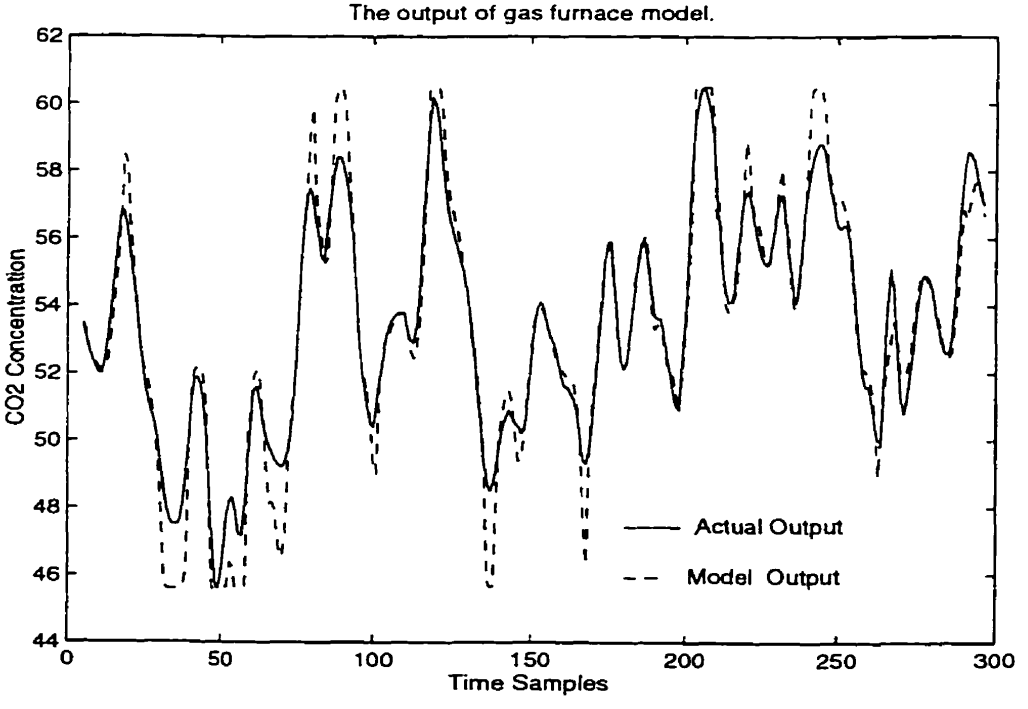


Figure 5.4 Output of the gas furnace model with MMFA rules.

Finally, the SGA algorithm (Section 5.2.4) is used to find the linguistic fuzzy rules of the gas furnace model. The algorithm is executed using a population size of 150 to decrease the chances of pre-mature convergence. Also, It is applied many times with different crossover and mutation probabilities, to obtain the best possible result. A satisfactory performance of the SGA is reached when the probability of crossover and mutation are set to 0.9 and 0.01, respectively. Out of the possible 49 rules of the fuzzy model, 37 rules are only considered. The rest are deleted because of their negligible effect (they have less than 5% of the highest firing rate of all the rules). The 37 rules are shown in Table 5.5. The model is simulated as shown in Figure 5.5, and its MSE value is found to be 0.6492 with the convergence rate shown in Figure 5.6.

Table 5.5 The complete fuzzy associative memory matrix with SGA rules.

	u(k-4)						
y(t-1)	NL	NM	NS	ZE	PS	PM	PL
NL				*NS*	*NM*	*NM*	*NM*
NM			*NVS*	*NS*	*NVS*	*NS*	*NM*
NS		*NVS*	*ZE*	*ZE*	*NVS*	*NVS*	*NS*
ZE	*NVS*	*PS*	*ZE*	*ZE*	*NVS*	*NS*	
PS	*PM*	*PVS*	*PVS*	*ZE*	*PVS*	*NVS*	
PM	*PS*	*PS*	*PVS*	*PVS*	*PS*		
PL	*PL*	*PS*	*PM*	*PM*	*PM*		

All entries correspond to $\dot{y}(t)$

5.4 Conclusions

This chapter presents four techniques to extract linguistic modeling/control rules from the input-output data of any plant. All the algorithms are implemented using the C++ language on a Pentium 166MHz. The techniques are tested using a well-known benchmark. The comparison among the different approaches is shown in Table 5.6. The CLA technique compared to the other techniques has a relatively low performance. Moreover, it has a relatively short execution time but not the shortest. The performance of the MDA algorithm is better than that of the CLA algorithm. However, the main disadvantage of this algorithm is finding the η_c value that gives the most appropriate set of rules as discussed before in Section 5.2.2. For this reason it is not as fast as the CLA and MMFA algorithms. The MMFA performance exceeds those of CLA and MDA and its execution time is also the shortest (among the four techniques).

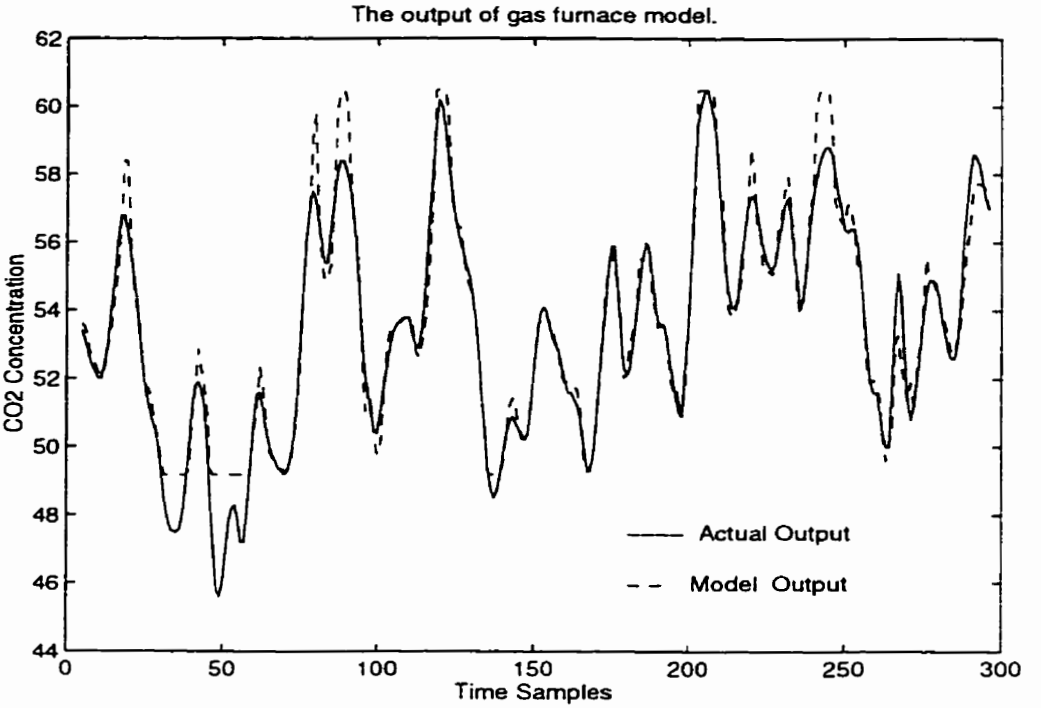


Figure 5.5 Output of the gas furnace model with SGA rules.

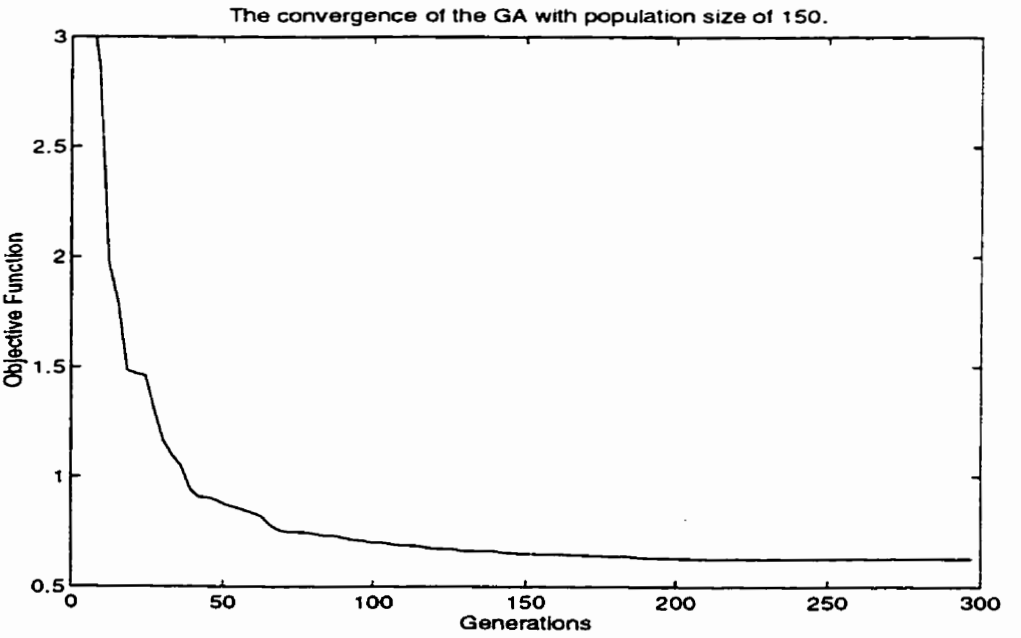


Figure 5.6 The SGA convergence rate with pcross=0.9 and pmut=0.01.

The SGA technique has the best performance but, at the same time, is much slower than the others. The SGA algorithm has a difficulty of estimating the pop_size, pcross, and pmut parameters that give the most appropriate performance. The CLA, MDA, and MMFA techniques are unsupervised learning algorithms while the SGA technique is a kind of supervised learning algorithm. For this reason the SGA algorithm is considered more robust than the other algorithms as it takes a feed-back guiding signal from the outside world, and it is expected to have the best performance and longest execution time.

The main performance difference among the four techniques occurs in the regions with low density of training data. In the regions with high training data density, they produced almost the same rules. This gives an indication about the behavior of these algorithms if a lack of and/or imprecise information exists.

The SGA is found to be the most suitable technique if the performance and the accuracy are the major concerns of the designer, especially if all these techniques run off-line. Also, the MMFA algorithm is the most suitable one if the performance is required with high speed, for instance, on-line and real-time applications. In other words, the MMFA has the best performance per unit computational cost ratio.

Table 5.6 The comparison among the different techniques.

Algorithm Name	No. of Rules	MSE	Computation Time	Learning Mode
CLA	28	2.1717	10.12 sec.	Unsupervised
MDA	34	0.9707	19.87 sec.	Unsupervised
MMFA	37	0.9441	3.17 sec.	Unsupervised
SGA	37	0.6492	30 min.	Supervised

Chapter 6

Optimization Phase

6.1 Introduction

In Chapter 4, we introduced the coarse identification phase of the hybrid learning scheme. Using this learning phase, we can find the most appropriate fuzzy partitioning for each input/output variable, the input/output scaling factors, and the initial parameters of the membership functions.

In Chapter 5, the rule-formation phase of the proposed learning scheme is presented. Using one of the techniques proposed in this phase, the linguistic "If-Then" rules could be extracted from both the input/output data and the expert knowledge.

In this chapter, the initial parameters of the membership functions, that are found in the coarse identification phase, will be optimized to give the best performance of the fuzzy model/controller according to a certain assessment criterion. Two different algorithms are described here to perform the optimization task. These techniques are: the Back-Propagation (BP) algorithm and the Multi-Resolutional Dynamic Genetic Algorithm (MRD-GA). A well-known benchmark is used to test and evaluate the two techniques. A comparative study between the two techniques is also presented. Conclusions are drawn from the assessment of each technique.

6.2 Back-propagation algorithm

The back-propagation training algorithm [96] is a generalization of the Least Mean Squares (LMS) algorithm. It uses a gradient-descent search technique to minimize a cost function equal to the mean square difference between the desired and the actual

network outputs. The network is trained by presenting all training data repeatedly. The network parameters are adjusted after trial using side information specifying the correct class until parameters converge and the cost function is reduced to an acceptable value. An essential component of the algorithm is the iterative method that propagates error terms required to adapt parameters back from nodes in the output layer to nodes in lower layers.

The goal of this supervised learning algorithm (refer to Figure 3.1) is to minimize the error function

$$E = \frac{1}{2}(y(t) - y^*(t))^2 \tag{6.1}$$

where $y(t)$ is the desired output, and $y^*(t)$ is the current output. The learning rules of each layer can be driven as follows:

Layer 5: The error signal of the output node is

$$\delta_1^5 = (y(t) - y^*(t)) \tag{6.2}$$

The mean (center) and the variance (width) of each output membership function are adapted by

$$m_i^4(k+1) = m_i^4(k) + \eta\delta_1^5 \frac{\sigma_i^4 O_i^4}{\sum_{j=1}^{n_3} \sigma_j^4 O_j^4} \tag{6.3}$$

$$\sigma_i^4(k+1) = \sigma_i^4(k) + \eta\delta_1^5 \frac{m_i^4 O_i^4 \left(\sum_{j=1}^{n_3} \sigma_j^4 O_j^4\right) - O_i^4 \left(\sum_{j=1}^{n_3} m_j^4 \sigma_j^4 O_j^4\right)}{\left(\sum_{j=1}^{n_3} \sigma_j^4 O_j^4\right)^2} \tag{6.4}$$

for $i = 1,2, ..., n_3$

where η is the learning rate of the network, and it is selected according to the problem and the experience of the designer, and it is preferred to be small and less than one.

Layer 4: The error signal of each node is

$$\delta_i^4 = \delta_1^5 \frac{m_i^4 \sigma_i^4 \left(\sum_{j=1}^{n_3} \sigma_j^4 O_j^4 \right) - \sigma_i^4 \left(\sum_{j=1}^{n_3} m_j^4 \sigma_j^4 O_j^4 \right)}{\left(\sum_{j=1}^{n_3} \sigma_j^4 O_j^4 \right)^2} \tag{6.5}$$

for i = 1,2, ..., n_3

Layer 3: No parameter needs to be adjusted in this layer, and only the error signal needs to be computed and propagated backward. That is,

$$\delta_i^3 = \sum_{j=1}^{n_3} w_{ij} \delta_j^4 \tag{6.6}$$

for i = 1,2, ..., $n_1 \times n_2$.

Layer 2: The mean and the variance of the input membership functions can be updated by

$$m_i^2(k+1) = m_i^2(k) - \eta \frac{\partial E}{\partial O_i^2} O_i^2 \frac{2(O_1^1 - m_i^2)}{\left(\sigma_i^2\right)^2} \tag{6.7}$$

$$\sigma_i^2(k+1) = \sigma_i^2(k) - \eta \frac{\partial E}{\partial O_i^2} O_i^2 \frac{2(O_1^1 - m_i^2)^2}{\left(\sigma_i^2\right)^3} \tag{6.8}$$

for i = 1,2,3, ..., n_1+n_2

where $$\frac{\partial E}{\partial O_i^2} = \sum_k q_k \tag{6.9}$$

where the summation is performed over the rule nodes that O_i^2 feeds into, and

$$q_k = \begin{cases} \delta_k^3, & \text{if } O_i^2 \text{ is minimum in kth rule node's input} \\ 0, & \text{otherwise} \end{cases} \tag{6.10}$$

After the error function converges to the desired error value, the resultant centers and widths represent the optimized membership functions of the fuzzy model.

6.3 Multi-resolutional dynamic genetic algorithm

A new approach is proposed here to use GAs for the optimization of the membership functions parameters. Problem-specific knowledge is used to tailor GAs to the needs of this learning phase. The main attribute of the proposed approach is that the fuzzy-model configuration is dynamically adapted while the optimization process is running. Accordingly, the proposed Multi-Resolutional Dynamic Genetic Algorithm (MRD-GA) changes its search space with the change of the problem configuration and with the advance of generations. The MRD-GA search space monotonically gets narrower and narrower, while the model parameters get closer and closer to the optimal values. Figure 6.1 illustrates this attribute by showing the behavior of one of the model parameters (a center of a membership function) during the optimization process. The ellipses in the figure represent the search space and the centers of these ellipses represent the different values taken by the center of the membership function. The figure shows how the resolution of the search space increases with the advance of generations, as the number of search points is kept constant and the search area gets smaller and smaller.

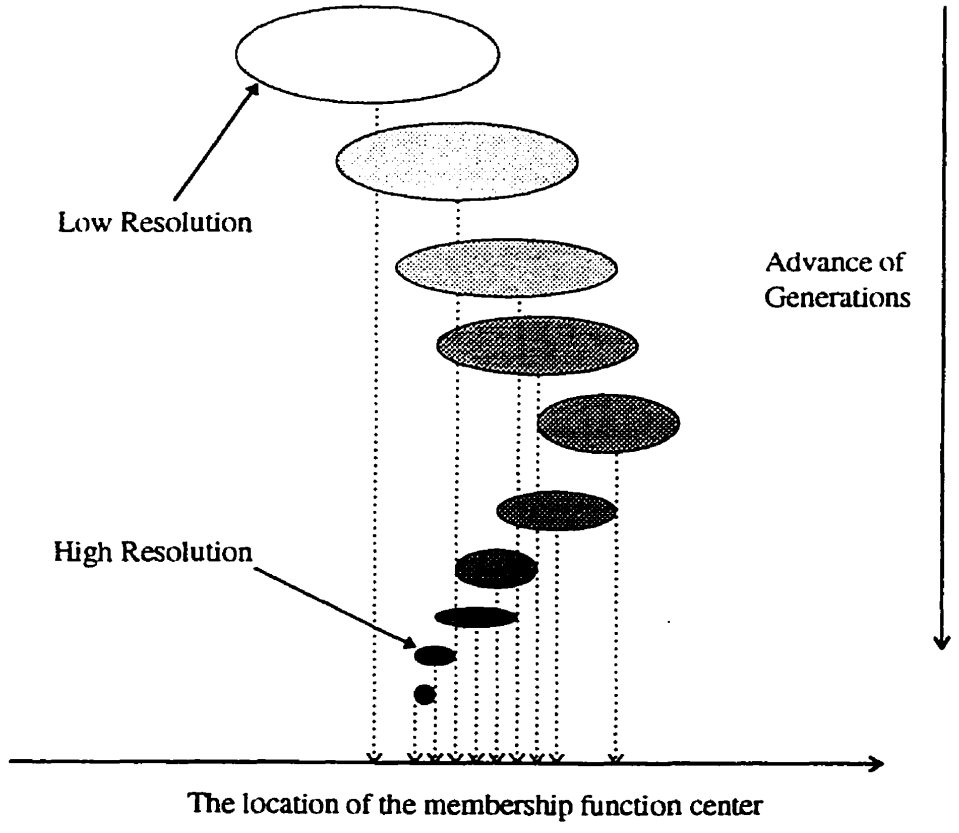


Figure 6.1 The adaptation process in the MRD-GA.

The MRD-GA is coded using the well-structured language C++. The program allows the user to define the values for population size (pop_size), maximum number of generations (max_gen), probability of crossover (pcross), and probability of mutation (pmut). In order to select the individuals for the next generation, the tournament selection method is used. In this method, two members of the population are selected at random and their fitness values compared. The member with higher fitness advances to the next generation. An advantage of this method is that it needs less computational effort than other methods. Also, it does not need a scaling process (like the roulette wheel selection). However, the particulars of the reproduction scheme are not critical to the performance of the MRD-GA; virtually, any reproduction scheme that biases the population toward the fitter strings works well [53].

The MRD-GA uses decimal-integer strings to encode the model parameters. The decimal strings are considered a more suitable representative method than the binary strings. This representation allows the use of more compact-size strings. The number of alleles (individual locations which make up the string) is determined from the total number of fuzzy sets used to partition the spaces of the input/output variables. For the model configuration shown in Figure 3.1, we have ($n_4=n_1+n_2+n_3$) membership functions. Each bell-shaped membership function is defined by two parameters (the center m_i and the width σ). To optimize the membership functions, we have to optimize ($n_4 \times 2$) parameters. Thus, the MRD-GA uses strings of length $n_4 \times 2$ alleles. It is allowed for each allele to take any value in the set [1, 2, ..., 9]. To convert the allele-value to a new center or width of a certain membership function, we use the following procedure:

Step 1: The initial values of the centers and widths of the fuzzy controller are entered to the GA program. for example, (m_{io}| i=1,2, ..., n_4) and (σ_{io}| i=1,2, ..., n_4).

Step 2: The new centers and widths are calculated from the allele values as

$$m_i = m_{io} + (s_i - 5)*\delta_m \qquad (6.11)$$

$$\sigma_i = \sigma_{io} + (s_{(i+n4)} - 5)*\delta_\sigma \qquad (6.12)$$

where m_i and σ_i are the new center and width values, respectively, s_i is the value of the i-th allele in the string, and δ_m and δ_σ are the offsets of the centers and widths, respectively. It is recommended to set these offsets to very small values (around 0.001). This allows a more stable convergence of the MRD-GA.

Step 3: If the allele value s_i of any center or width equals '5' then no change occurs. If s_i is greater than '5' then a positive change occurs (the center or width increases). If it is less than '5' then a negative change occurs (the center or width decreases).

The MRD-GA uses the Mean Squared Error (MSE) (the error is the difference between the actual output and the estimated output by the fuzzy model) as a fitness function. Simply, for each chromosome (1/MSE) is considered as the fitness measure of it. The MSE is calculated from N data points as

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (u_i - \hat{u}_i)^2 \tag{6.13}$$

where u_i is the actual value, and $\hat{u}_i$ is the estimated value.

The MRD-GA can also use a fitness function that has an inverse proportionality with some of controller performance indices such as the settling time, overshoot, and integral of error. An example of such fitness function is

$$f = \frac{1}{k_1 \int e(t)^2 dt + k_2 T_s} \tag{6.14}$$

where f is the fitness function, $e(t)$ is the controller output error at time t, T_s is the settling time, and k_1 and k_2 are weighting factors.

Each τ generations, the offset values (δ_m and δ_σ) decrease according to the following decaying functions:

$$\delta_m = \delta_m \times \theta_m \qquad 0 < \theta_m < 1 \tag{6.15}$$

$$\delta_\sigma = \delta_\sigma \times \theta_\sigma \qquad 0 < \theta_\sigma < 1 \tag{6.16}$$

where θ_m and θ_σ are the modifying factors for the centers and widths, respectively. The decaying functions can take any decaying shape such as an exponential decay. The usual GA terminating condition is a maximum allowable number of generations or a certain value of MSE required to be reached. In this GA algorithm, the stopping criterion is the execution of a certain number of generations without any improvement in the best fitness value. In this criterion, you do not need to specify a required MSE value (which is usually unknown in advance) or a required number of generations (where there is no guarantee that this number will produce an appropriate solution).

The MRD-GA pseudocode is:

Initialize P(t)→P(0). *:P(t) Population at time t.*
Initialize best_fit = 0. *: The best fitness value.*
Evaluate P(0).
Search for the best fitness of P(0) and assign best_fit to it.
While (not terminate-condition) do
Begin
t← t+1 *:Increment generation.*
If mod(t/τ)=0 then modify (decrease) δ_m *and* δ_σ *as given in (6.15), and (6.16).*
Select P(t) from P(t-1) using tournament selection criteria.
Recombine P(t) :apply genetic operators (crossover, mutation).
Evaluate P(t).
Search for the best fitness of P(t) and compare it with best_fit. if larger then do
Begin
Assign best_fit to the best fitness value of P(t).
Adapt the centers and the widths ($(m_{io} | i=1,2, ..., n_4)$ *and* ($\sigma_{io} | i=1,2, ..., n_4$)) *according to the state of the chromosome having the best fitness using (6.11), and (6.12).*
End
End
End.

The above GA offers exciting advantages over the conventional GA [105-107] (the conventional GA is like the one described in Section 2.4, which its parameters are kept constant during the optimization process (i.e. static GA)). The MRD-GA allows a dynamic increase in the resolution of the search space (by decreasing δ_m and δ_σ) as the model parameters approach their optimal values. It also changes the nature of the model-identification problem from a static type to a dynamic type (by adapting m_{io} & σ_{io} continuously) which decreases the chances of the GA premature convergence, as this dynamic feature preserves the diversity within the GA's populations.

6.4 Testing and evaluation study

The benchmark used in this section is the one used in the previous chapter (the gas furnace model [104]). The initial parameters of the input/output membership functions are found by SOM and shown in Figure 5.1. The MMFA algorithm is used to find the 37 linguistic rules that are shown in Figure 5.4. Both BP algorithm and MRD-GA algorithm will be used to optimize the parameters of the membership functions, in order to build an accurate linguistic fuzzy model for the gas furnace.

The BP algorithm is applied with different learning rates (η). It is found that, using the gas furnace data, the BP algorithm gets trapped in local minima except with very small learning rates (0.001~0.005), and also suffers from divergence problems using large learning rates (0.5~2). Figure 6.2 shows the convergence curves of BP algorithm using various learning rates.

The MRD-GA is then applied to optimize the membership function parameters. This algorithm has many control parameters which should be selected before running, such as the population size (pop_size), crossover probability (pcross), type of the crossover operator, mutation probability (pmut), the offset values (δ_m and δ_σ), the modifying factors (θ_m and θ_σ), and the decaying time constant (τ). The last five parameters are mainly used to convert the genetic algorithm from an integer optimization technique to a continuos optimization one. The effect of the first three parameters on the performance of the MRD-GA will be studied, as the algorithm is more sensitive to the variation in these parameters than the others.

To study the effect of the pop_size, the MRD-GA parameters are set as follows: pcross = 0.9, pmut = 0.1, chromosome-length = 46, $\delta_m = 0.0006$, $\delta_\sigma = 0.00025$, $\theta_m = 0.99$, $\theta_\sigma = 0.99$, and $\tau = 10$. To study the pop_size effect on the search performance of the proposed MRD-GA, the MRD-GA is applied four times with different population sizes (pop_size = 80, 50, 30, and 12). After finishing the second learning phase and before applying the MRD-GA, the model has an MSE value of 0.9441. This MSE value is decreased to 0.111 after 4972 generations of the MRD-GA using pop_size = 50 (note that the MSE value reached 0.15 after only 900 generations). The MSE decay rates using different population sizes are shown in Figure 6.3.

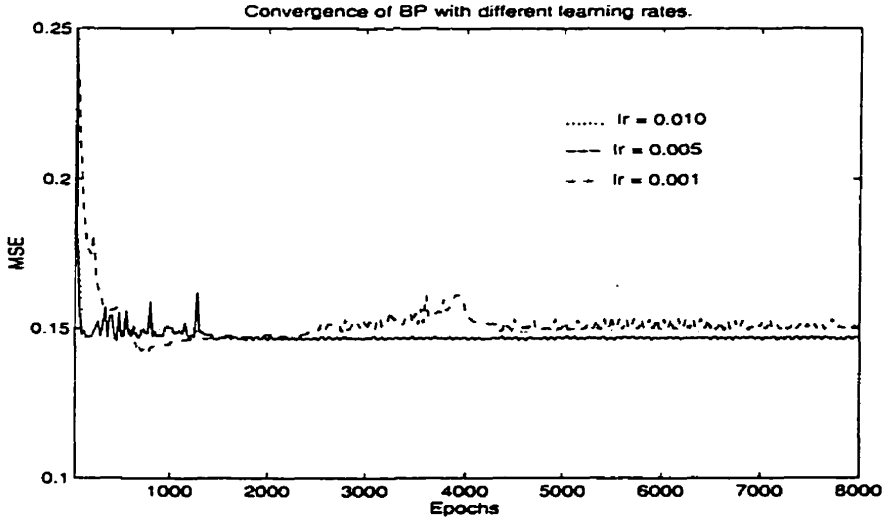


Figure 6.2 Convergence curves of back-propagation algorithm.

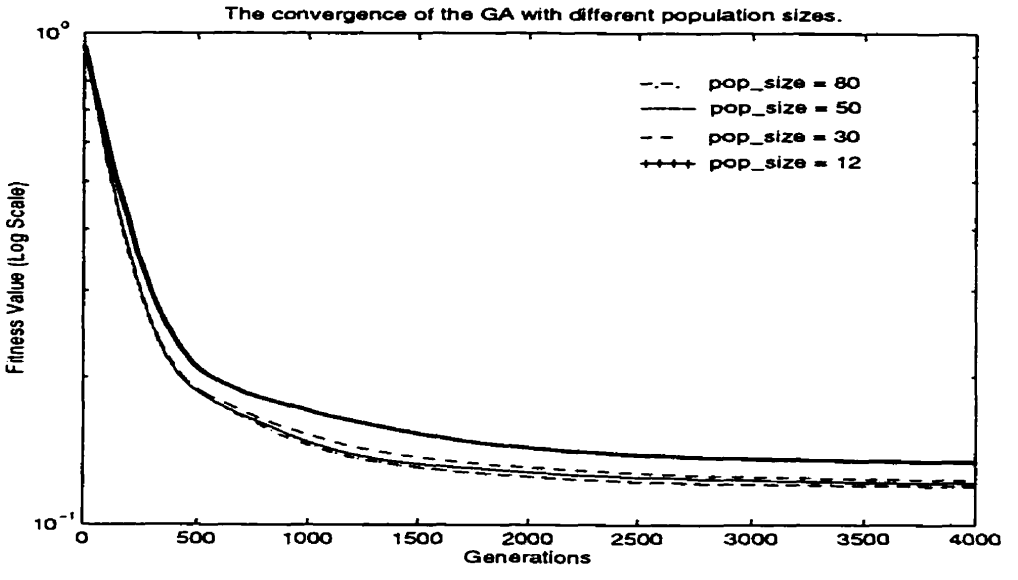


Figure 6.3 The MRD-GA convergence rates with different population sizes.

To compare and evaluate the performance of the MRD-GA that uses different population sizes, an MSE value of 0.14 is selected to be the stopping criteria. The results in Table 6.1 show that the increase in the population size decreases the number of generations needed to attain this MSE, and increases the accuracy. However, we are more concerned about the computation time, which is directly proportional to the population size for the same number of generations. The population size of 12 requires less computation than the other population sizes to reach an MSE of 0.14, but the minimum MSE attained is larger than the others. A compromise between the accuracy and the computation amount should be done in selecting the appropriate population size. From the results in Table 6.1, it is obvious that there is no big difference in the accuracy for 80, 50, and 30 population sizes. Accordingly, it is recognized that the proposed MRD-GA has less sensitivity to the population size than the conventional GA due to its dynamic feature. Therefore, population sizes around 30 are considered suitable for this modeling example.

Table 6.1 Convergence speed at different population sizes.

Population Size	No. of Generations for MSE = 0.14	minimum MSE (4000 Gen.)
80	1119	0.1192
50	1176	0.1206
30	1355	0.1223
12	2338	0.1333

To study the effect of the crossover probability on the performance of the MRD-GA, The algorithm is applied to optimize the parameters of a second order dynamic-system (which will be explained in details in Section 7.3). The algorithm parameters are set as follows: pop_size = 20, pmut = 0.05, chromosome-length = 48, δ_m = 0.0001, δ_σ = 0.0001, θ_m = 0.99, θ_σ = 0.99, and τ = 25. The MRD-GA is applied many times with different crossover probabilities. After finishing the second learning phase and before applying the MRD-GA, the model has an MSE value of 0.2058. This MSE value is decreased to 0.0374 after 5533 MRD-GA generations using a single point crossover with pcross value of 0.25. The MSE decay rates using different crossover probabilities are shown in Figure 6.4. To compare and evaluate the performance of the MRD-GA that uses different crossover probabilities, an MSE value of 0.041 is selected to be the stopping criterion. The results in Table 6.2 show the effect of pcross on the convergence rate of the MRD-GA. Each entry in this table is the average of three individual runs. A low pcross value (0.1~0.3) limits the MRD-GA from reproducing fitter individuals through the advance of generations, while a high pcross value (0.95~1.0) has a detrimental effect on the diversity of the population which may cause a premature convergence. The previous two factors have a considerable effect on the search performance of the MRD-GA. Table 6.2 shows that the search performance has two peaks at pcross values of 0.6 and 0.9 respectively. At these values, the global (overall) effect of the two factors are minimum. Accordingly, the intermediate range of pcross (0.5~0.9) is considered, in this example, suitable for the proposed MRD-GA.

Table 6.2 The convergence speed of different crossover rates.

Probability of crossover	No. of Generations for MSE = 0.041
0.10	4201
0.25	2807
0.50	2545
0.60	2328
0.75	3175
0.85	2659
0.90	1784
0.99	2331

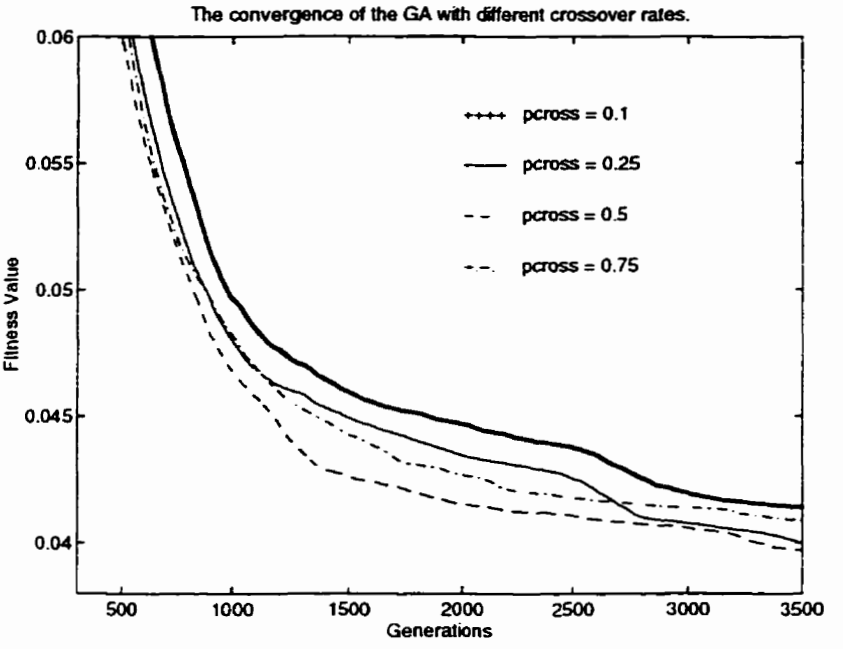


Figure 6.4 MRD-GA convergence curves with different crossover rates.

The performance of the proposed MRD-GA is then investigated using a different type of crossover operators (2-point crossover). The 2-point crossover operator uses two randomly selected sites instead of one to perform the crossover operation. For example, if we have two chromosomes; [a_1 a_2 a_3 a_4 a_5 a_6], [b_1 b_2 b_3 b_4 b_5 b_6] and the selected two sites are 2 and 5. The resultant offspring chromosomes will be [a_1 a_2 b_3 b_4 b_5 a_6], [b_1 b_2 a_3 a_4 a_5 b_6]. Table 6.3 shows that the 2-point crossover operator has degraded the performance of the algorithm. This result sustains the results of De Jong's study [108]. An intuitive explanation for this observation can be found by counting the number of unique operators involved [31]. In the case of simple crossover, we have not just a single operator but a set of l-1 (where l is the string length) operators. With a 2-point crossover there are $l(l-1)/2$ different ways of picking the two crossover points. As a result, each operator is less likely to be preserved. With more mixing and less structure, these more involved crossover operators become like a random shuffle and fewer important schemata can be preserved, especially for short strings. This also explains the good performance of the 2-point crossover at high crossover rates (0.99), at which some randomization is needed to preserve the population diversity.

Table 6.3 Comparison between 1-point crossover and 2-point crossover.

	No. of generations for MSE = 0.041	
Pcross value	1-point crossover	2-point crossover
0.25	2807	3024
0.60	2328	3289
0.90	1784	2396
0.99	2331	1951

6.5 Conclusions

This chapter presents two techniques to optimize the parameters of the membership functions of a fuzzy model. The BP and MRD-GA are implemented using the C++ language on a Pentium 166MHz, and then tested using the gas furnace benchmark. A comparative study of these approaches is shown in Table 6.4.

The table shows that the MRD-GA has better performance (better MSE). However, it takes more time to converge than the BP. Both of the two algorithms are supervised algorithms. The BP algorithm minimizes the MSE between the desired and the fuzzy model outputs. For this reason, it needs both input and output data samples to work. However, the input-output data is usually available in modeling applications but not in control applications, in which the output of the controller can not be specified in advance. On the contrary, the MRD-GA algorithm optimizes an objective function. This objective function can be tailored to the needs of the designer, without the necessity to obtain output data samples for the system under study. This feature makes the MRD-GA suitable for both modeling and control purposes as will be shown in Chapters 7 and 8.

One of the disadvantages of the MRD-GA is the existence of many control parameters that should be determined before applying the algorithm. A sensitivity study for some of these parameters is presented in Section 6.3, in order to give the reader a brief guide for the selection of these parameters. On the other hand, the BP algorithm has the advantage of having only one control parameter (η) or two if a momentum term is added to the algorithm.

The MRD-GA has also other advantages over the back-propagation (BP) algorithm. The MRD-GA allows one to obtain intermediate solutions, which the BP usually can't offer; also, the GA does not suffer from convergence problems with the

same degree that the BP suffers (i.e., the MRD-GA is more robust). The BP, especially in hardly nonlinear problems, is much more likely to get trapped in local minima than the MRD-GA, which is more likely to converge toward a global solution. A justification for this feature is that the MRD-GA has parallel processing and dynamic properties that prevent premature convergence to occur and simultaneously offer various solution candidates for the optimization problem in every generation. On the other hand, BP algorithm does not have these features and its convergence uses a successive linearization procedure with step sizes dependent on the used learning rate and/or momentum factor.

Some sensitivity analyses have been done to study the effect of some control parameters on the performance of the MRD-GA. Of course, the type of data used in this analysis has an undeniable effect on the results of this study. However, the main purpose is just to give a quick guide for the user to select appropriate values for the control parameters. From the study, it is found that the proposed algorithm is less sensitive to the change of the pop_size than the conventional (static) GA [106], and pop_size values in the range of 20~50 are considered appropriate for most of the applications. Also, pcross values in the range of 0.5~0.9 are considered suitable for this algorithm. It is also noticed from the study that the 1-point crossover operator gives better performance than the 2-point crossover operator does. This is due to the fact that the 2-point crossover produces an excessive random shuffling for the chromosomes in an algorithm that does not need more sources of population diversity, which implicitly exists due to the dynamic feature of this algorithm.

Table 6.4 Comparative study between the BP and MRD-GA algorithms.

View Point	Back-Propagation	MRD-GA
Benchmark	Gas Furnace Data	Gas Furnace Data
Best MSE	0.143	0.111
Iterations	20,000 epochs	4,972 generations
Computation Time	49 min.	124 min.
Programming Environment	C++	C++
Computing Environment	Pentium 166MHz	Pentium 166MHz
Learning Mode	Supervised, to minimize the MSE	Supervised, to optimize an objective function
No. of control parameters	Few (like η)	Many (like pop_size, pcross, and pmut)
Convergence	Subject to get trapped in local minima	Clever at escaping from local minima
Applicability	Less flexibility	More flexibility

Chapter 7

Intelligent Modeling of Complex Systems

7.1 Introduction

Linguistic modeling of complex irregular systems constitutes the heart of many control and decision-making systems, and fuzzy logic represents one of the most effective algorithms to build such linguistic models. In this chapter, two well-known benchmarks are modeled using the proposed hybrid learning scheme. The performances of the models built for the two benchmarks provide a full assessment for the proposed intelligent hybrid system. Moreover, to show the effectiveness of the proposed approach, this approach is compared with other intelligent modeling approaches.

7.2 Gas furnace model

The benchmark used in this section is the one used in Chapters 5 & 6 (the gas furnace model [104]). The gas furnace is modeled using the fuzzy-neural network shown in Figure 3.1. The initial parameters of the input/output membership functions are found by SOM and are shown in Figure 5.1. The MMFA algorithm is used to find the 37 linguistic rules that are shown in Figure 5.4. The MRD-GA algorithm (Section 6.4) is then used to optimize the parameters of the membership functions, in order to build an accurate linguistic fuzzy model for the gas furnace.

The computation time elapsed to perform the whole learning scheme is roughly determined as shown in Table 7.1. The resultant membership functions and the model output are shown in Figure 7.1 and 7.2, respectively. The MSE decrease rates using different population sizes are also shown in Figure 6.3.

Table 7.1 Computation time of the gas furnace model.

Implementation	C++ codes on a Pentium 166 MHz.		
Computation Time	Phase I	Phase II	Phase III
	2 sec.	3 sec.	124 min.
Learning Mode	Unsupervised (9000 epochs)	Unsupervised (37 rules)	Supervised (4972 generations)

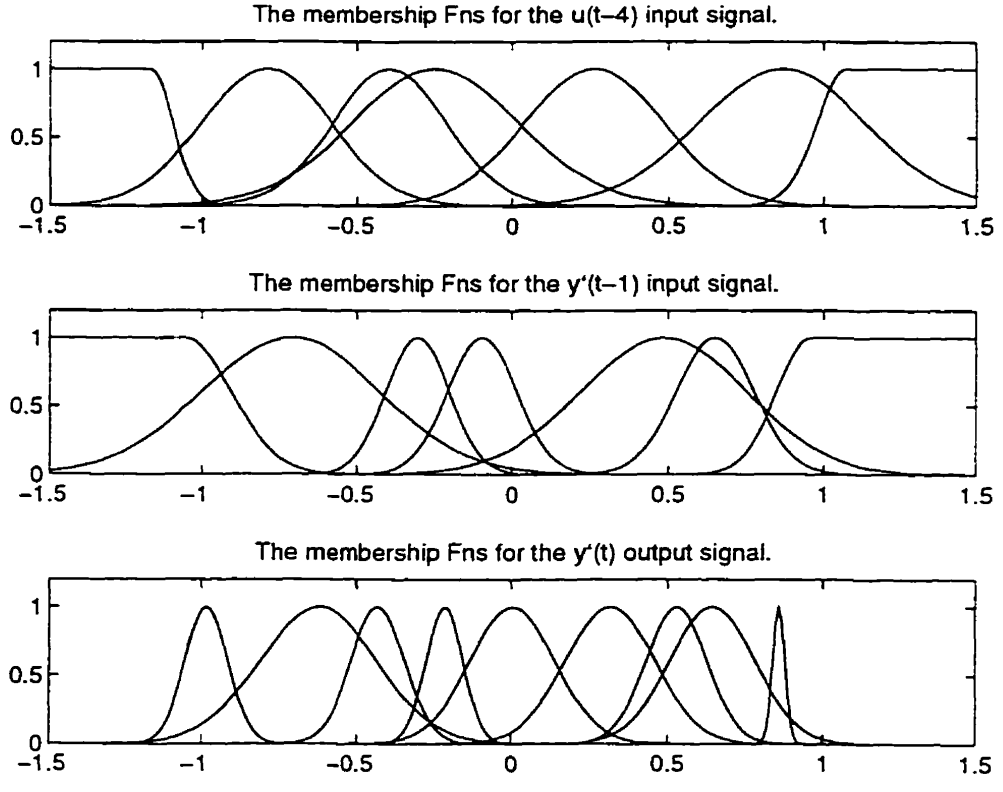


Figure 7.1 The optimized membership functions after the MRD-GA.

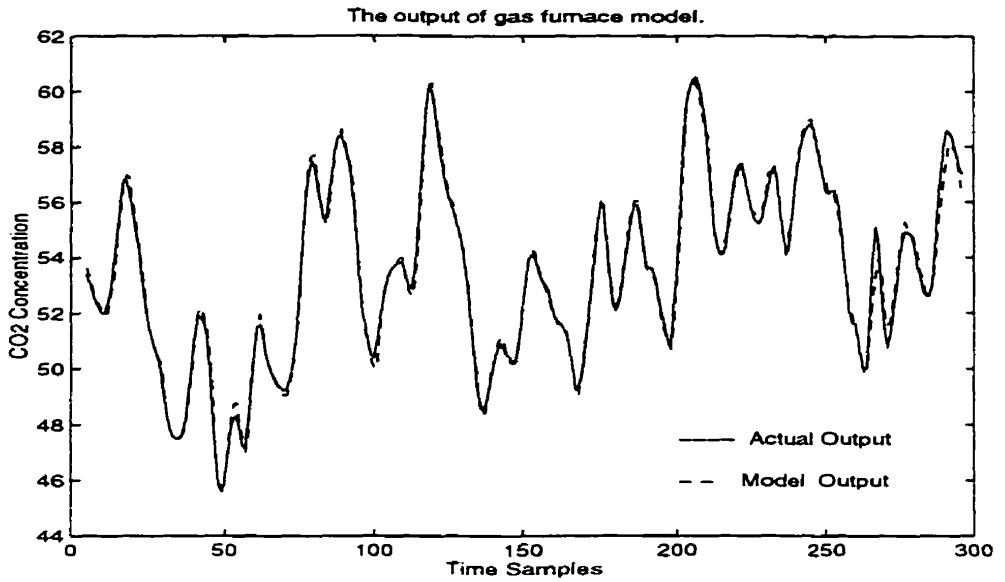


Figure 7.2 Output of the gas furnace fuzzy model.

In Table 7.2, our fuzzy model is compared with other models identified from the same data. It can be seen that our model outperforms all the other models in its class (class B, refer to equation 3.2). In comparison with class-A models (refer to equation 3.1), *Sugeno's* model [79] has less MSE value using six inputs but, at the same time, has much higher MSE value using the same inputs used by our model (*y(k-1), and u(k-4)).* Also, this model is quite difficult to build [77, 80, 81]; the most difficult aspect lies in the identification of the premise structure, mainly the membership functions of the input variables. For each membership function, at least two or three parameters have to be calculated through a nonlinear programming procedure. The choice and computation of these membership functions are rather tricky and subjective so that it is possible for different designers to sometimes get completely different results.

Wang's model (class A) [77] has comparable results and less number of rules; however, the number of rules does not necessarily give a reliable indication of the number of unknown parameters of the model. For example, in Wang's model [77] five class-A rules are used with two inputs, the number of unknown parameters in this case (in both the premise and consequent parts) are 35. In our model, 37 class B rules are used with two inputs and 46 unknown parameters. Bearing in mind that our model shows about 30% decline in the MSE value and provides a linguistic description for the gas furnace system, these two advantages, in our view, compensate for the difference in the number of parameters (46 versus 35).

Table 7.2 Comparison of our model with other models.

Model Name	Type	Inputs	Number of Rules	MSE
Tong's model [109]	Fuzzy, Class B	y_{k-1} u_{k-4}	19	0.469
Pedrycz's model [74]	Fuzzy, Class B	y_{k-1} u_{k-4}	81	0.320
Xu's model [75]	Fuzzy, Class B	y_{k-1} u_{k-4}	25	0.328
Box's model [104]	Linear	y_{k-1} y_{k-2} u_{k-3} u_{k-4} u_{k-5}	—	0.202
Sugeno's model [79]	Fuzzy, Class A	y_{k-1} y_{k-2} y_{k-3} u_{k-3} u_{k-4} u_{k-5}	2	0.068
Sugeno's model [79]	Fuzzy, Class A	y_{k-1} u_{k-4}	2	0.359
Sugeno's model [76]	Fuzzy, Class B	y_{k-1} u_{k-3} u_{k-4}	6	0.190
Wang's model [77]	Fuzzy, Class A	y_{k-1} u_{k-4}	5	0.158
Our model	Fuzzy, Class B	y_{k-1} u_{k-4}	37	0.111

7.3 Second-order system's model

This example is taken from *Narendra et al.* [22] in which the plant to be identified is given by the second-order highly nonlinear difference equation

$$y_k = \frac{y_{k-1} y_{k-2} (y_{k-1} + 2.5)}{1 + y_{k-1}^2 + y_{k-2}^2} + u_k \qquad (7.1)$$

Training data of 500 points are generated from the plant model by assuming a random input signal 'u_k' uniformly distributed in the interval [-2, 2]. This data is used to build a linguistic-fuzzy model for this plant.

The plant is modeled using the FNN described in Section 3.2. The model has three inputs u_k, y_{k-1}, and y_{k-2}, and a single output y_k. The inputs u_k and y_{k-1} are intuitively partitioned into five fuzzy linguistic spaces {NL, NS, ZE, PS, PL}, the input y_{k-2} is partitioned into three fuzzy spaces {N, Z, P} and the output y_k is partitioned into eleven fuzzy spaces {NVL, NL, NM, NS, NVS, ZE, PVS, PS, PM, PL, PVL}. The SOM algorithm described in Section 4.2 is used to determine the initial centers and widths of the 24 membership functions of the input/output variables of the fuzzy model. The four scaling factors of this fuzzy model are determined from this learning phase as '$G_u = 0.7476$', '$G_y = 0.4727$', '$G_{yy} = 0.6261$', and '$G_o = 5.5781$'.

According to the structure of this fuzzy-neural network, the number of rules (rule-nodes in the third layer) is 5×5×3 = 75. The MMFA algorithm (Section 5.2.3) is used to find the 75 rules of this fuzzy model and the results are shown in Table 7.3.

The MRD-GA (Section 6.3) is applied to optimize the parameters of the dynamic-system model. The algorithm parameters are set as follows: pop_size = 20, pmut = 0.05, chromosome-length = 48, $\delta_m = 0.0001$, $\delta_\sigma = 0.0001$, $\theta_m = 0.99$, $\theta_\sigma = 0.99$, and $\tau = 25$. After finishing the second learning phase and before applying the MRD-GA, the model has an MSE value of 0.2058. This MSE value is decreased to 0.0374 after 3517 generations using a single point crossover with pcross value of 0.9 (note that the MSE value reached 0.06 after only 470 generations). The computation time used to perform this learning process is illustrated in Table 7.4. The MSE decay rates using different crossover probabilities are shown in Figure 6.4.

Table 7.3 The complete FAM matrices with the fuzzy rules.

y_{k-2}=N

u_k \ y_{k-1}	NL	NS	ZE	PS	PL
NL	NS	NM	NM	NVL	NVL
NS	NS	NS	NM	NL	NL
ZE	ZE	NVS	NS	NS	NS
PS	PVS	PVS	ZE	NVS	NVS
PL	PL	PM	PS	ZE	ZE

y_{k-2}=Z

u_k \ y_{k-1}	NL	NS	ZE	PS	PL
NL	NL	NM	NM	NM	NS
NS	NM	NM	NS	NVS	ZE
ZE	NVS	NVS	NVS	ZE	ZE
PS	PVS	ZE	PVS	PVS	PM
PL	PVS	PVS	PS	PM	PS

y_{k-2}=P

u_k \ y_{k-1}	NL	NS	ZE	PS	PL
NL	NL	NM	NS	NVS	PVS
NS	NS	NS	NVS	ZE	PS
ZE	NVS	NVS	ZE	PS	PM
PS	ZE	ZE	PS	PM	PL
PL	PVS	PVS	PS	PL	PVL

All entries correspond to y_k

Table 7.4 The computation time of the second order model.

Implementation	C++ codes on a Pentium 166 MHz.		
Computation Time	Phase I	Phase II	Phase III
	4 sec.	7 sec.	59 min.
Learning Mode	Unsupervised (12000 epochs)	Unsupervised (75 rules)	Supervised (3517 generations)

After the learning process is finished, the model is tested by applying a sinusoidal input signal $u_k = sin(2\pi k/25)$ to the fuzzy model. The output of both the fuzzy model and the actual model are shown in Figure 7.3. The fuzzy model has a good match with the actual model with an MSE of '0.0403'. Another test is carried out using an input signal $u_k = 1.6 \times cos(2\pi k/30)$. The result is shown in Figure 7.4 and the MSE in this

case is '0.0369'. After extensive testing and simulations, the fuzzy model proved an excellent performance in forecasting the output of the complex-dynamic plant. Remember that in this example only 500 data points are used to build the model; while in [22], 100,000 data points have been used to identify a neural network model. It can be expected that the performance of the identified fuzzy model may be further improved if the number of data points used to build the model is increased.

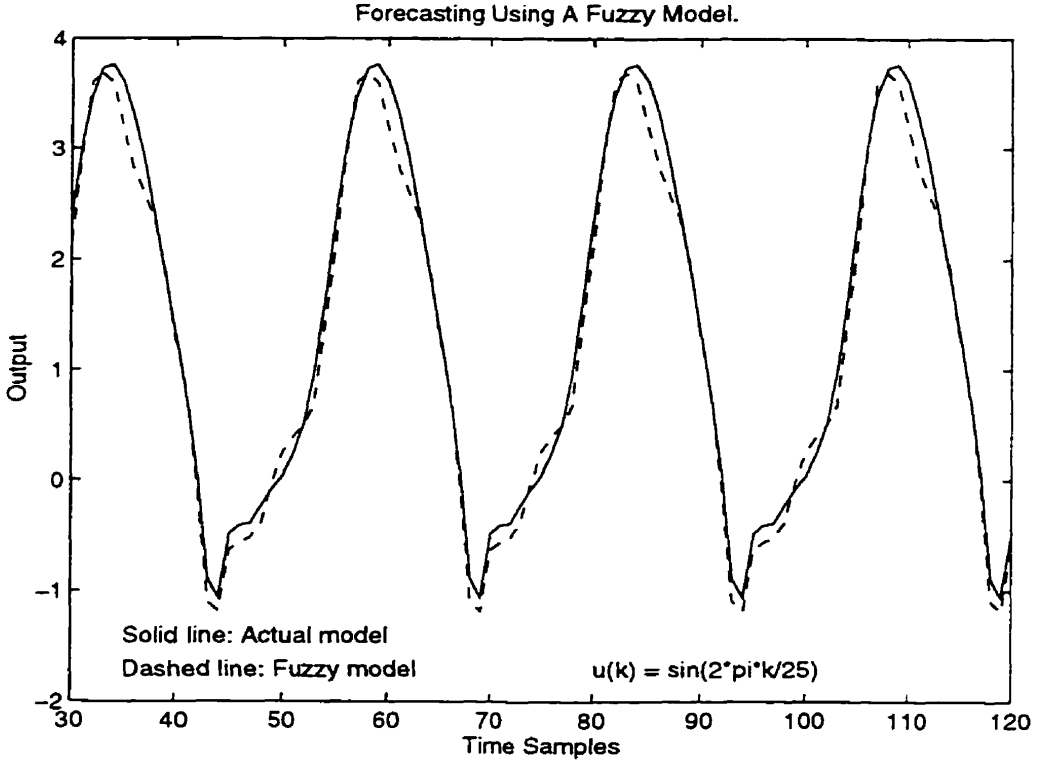


Figure 7.3 Testing of the fuzzy model vs. the actual model.

In order to compare our modeling approach with that of *Sugeno's* [19, 79] and *Wang's* [77] approaches, both of these approaches are implemented. The *Sugeno's* approach is implemented using the MATLAB fuzzy-logic toolbox. The approach [52] applies the Least-Squares Algorithm (LSA) and the back-propagation gradient descent method for identifying the linear (consequent) and nonlinear (premise) parameters of the class-A fuzzy rules, respectively. The core function of this algorithm is implemented using an-optimized-for-speed C code. Wang's approach is implemented using the C++ programming language. The approach uses the Fuzzy C-Means (FCM) clustering algorithm [89] to find the premise parameters of the class-A fuzzy rules, then applies the least squares algorithm to find the consequent linear parameters of the rules.

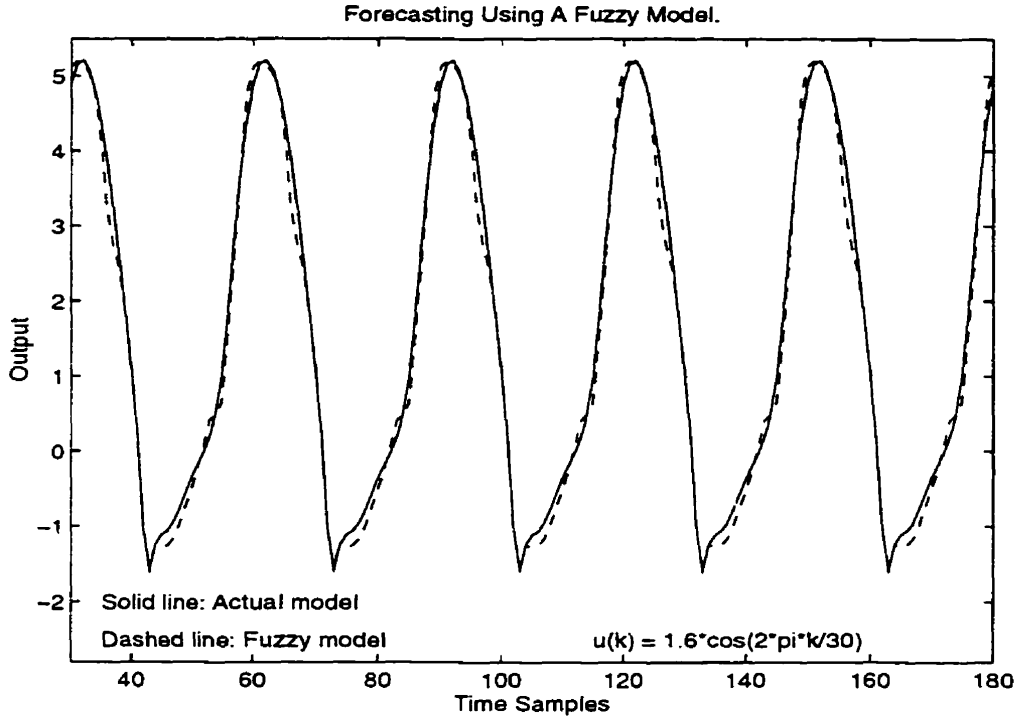


Figure 7.4 Testing of the fuzzy model vs. the actual model.

Table 7.5 compares our modeling approach with both of *Sugeno's* and *Wang's* approaches. The models are learned from the previously randomly generated 500 data pairs and tested by applying a sinusoidal input signal $u_k = sin(2\pi k/25)$. All the experiments are carried out on a Pentium 166MHz PC. The comparison shows the advantages of our modeling approach.

Table 7.5 A modeling comparison using the second order system.

View point	Our Model	*Sugeno's* Model	*Wang's* Model
Type	Class B	Class A	Class A
Input variables	*u(k), y(k-1), y(k-2)*	*u(k), y(k-1), y(k-2)*	*u(k), y(k-1), y(k-2)*
No. of rules	75	12	8
No. of linear parameters	——	48	32
No. of nonlinear parameters	48	14	48
Total no. of parameters	48	62	80
Learning MSE	0.0374	0.5072	0.6184
Testing MSE	0.0403	0.2447	0.2037
Computation time	59 min.	91 min.	19 min.

Chapter 8

Intelligent Control of Synchronous Machines

8.1 Introduction

The highly interconnected nature of power systems makes their operation and control complex processes. The disturbances in some elements may affect the whole system operation and stability, causing poor power quality or even interruption of power supply [110-112]. For analytical studies, researchers have classified the power system stability into three categories [110-114]:

1. *Steady-state stability*: This corresponds to the stability of a power system around an operating point. If the system is able to maintain synchronism after small changes in operating conditions, it is said that it has steady-state stability.

2. *Transient stability*: This refers to the ability of a power system to regain stability after a sudden and severe disturbance. System faults, line-switching, and large changes in loads can be considered as severe disturbances that may lead to transient stability problems.

3. *Dynamic stability*: It is the stability of a power system under small and sudden disturbances. These types of disturbances can lead to long-term sustained oscillations [113].

Many techniques have been proposed to control synchronous generators in order to overcome stability problems. One of the most basic techniques is to introduce damper windings in synchronous generators [111-112] to damp out the speed oscillations. Other methods include governor control [114-117], capacitor switching control [118-119] and excitation control [120-122]. Out of these methods, excitation control has

been given most attention because a synchronous generator excitation loop has a small time constant as compared to a mechanical governor time constant and, hence, fast response is expected.

Since the beginning of the late 1950's and early 1960's most of the generating units have been equipped with a continuously-acting Automatic Voltage Regulator (AVR) to improve the voltage profile at the consumer end and to enhance the power system transient stability. Power System Stabilizers (PSSs) have been used to enhance the performance of the AVRs. These stabilizers provide a supplementary control signal (in phase with the speed deviation of the rotor) to the excitation control loop [111, 123-124]. The whole system configuration is shown in Figure 8.1.

In the study of a single-machine infinite-bus system, the power system experiences only a single-mode of oscillation. This is not the case in real power systems in which a large number of synchronous generators, with quite different inertia constants, are connected together through transmission lines. In such systems, it is common to find two groups (or more) of generators that are weakly inter-connected. This results in multi-mode oscillation phenomenon, which is divided into three modes of oscillation [110]:

1. *Inter-machine mode*: which describes frequencies related to closely coupled generators that swing relative to each other. These frequencies are in the range of 0.8-2.0 Hz.

2. *Local mode*: usually refers to oscillations occurring in plant transients, caused by generator rotors that oscillate relative to the combined equivalent inertia of the whole plant. Local mode oscillation frequencies are in the range of 0.5-1.5 Hz.

3. *Inter-area modes*: these frequencies are caused by coherent groups of generators in one area which swing relative to a number of other coherent groups of generators in other areas. These frequencies are in the range of 0.1-0.7 Hz.

8.2 Review

Starting from the late 1950's, various types of techniques and control theories have been used for synchronous-machine control in order to enhance the stability of power systems. We categorize these techniques into three different groups as follows.

1. *Conventional control techniques*: These include Proportional-Integral-Derivative control (PID), lead-lag compensators and pole placement strategies.

2. *Modern control techniques*: These include linear optimal control, adaptive control, and H_∞ (robust) control strategies.

3. *Artificial intelligence techniques*: These include Expert Systems (ES), Neural Networks (NN), Fuzzy Logic (FL), and Hybrid-Fuzzy control strategies.

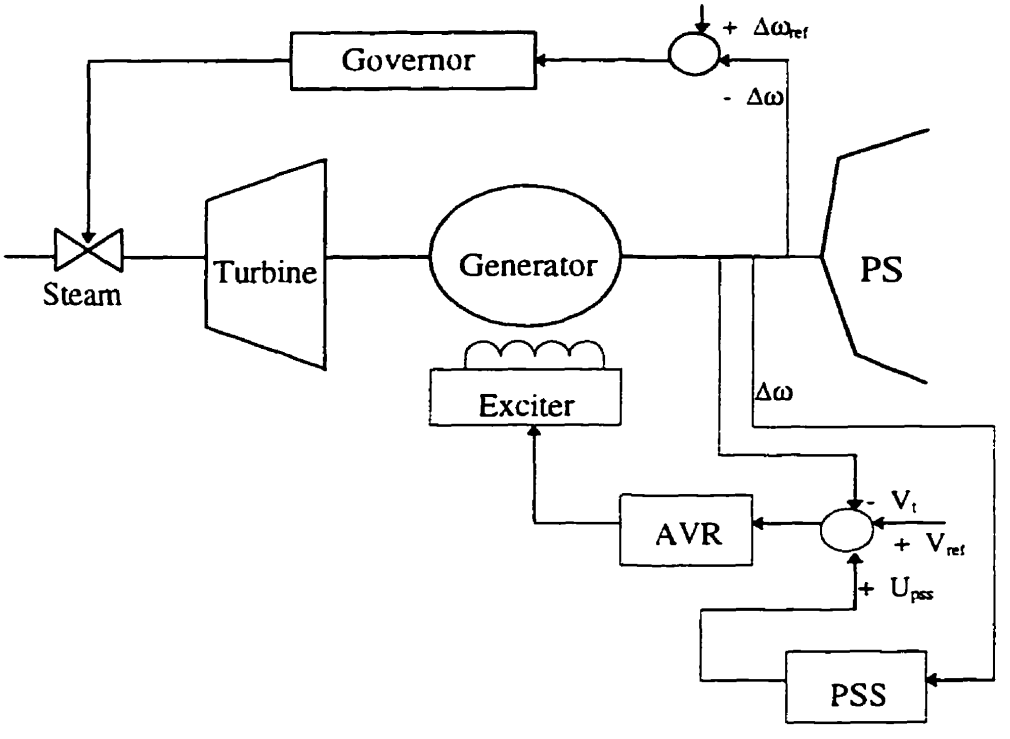


Figure 8.1 Block diagram of a generating unit including the AVR and PSS.

Designed using classical control theory, conventional PSSs (CPSSs) have been able to improve the stability limits of the system. However, their performance may deteriorate with changes of the operating point. This is because the conventional PSSs are designed using a linearized model of the machine at a prescribed operating point and, in practice, power systems are highly non-linear with stochastic operation in nature. For example, the gain of a plant increases with the generator loading [120]. Thus, controller parameters that are optimum for one set of operating conditions, may not be optimum for another set of operating conditions. This has opened the door for more research using modern control techniques.

The linear optimal control is one of the modern control techniques [125-127]. The main drawback of this technique is that it is based on a linearized system model that corresponds to a given operating point. Thus, it also has the same limitations that faced the CPSSs. Another alternative is the adaptive control, which has been found to suit the stochastic nature of power systems. The advantage of adaptive power system stabilizers (APSS) is their ability to adjust controller parameters on-line according to the current operating conditions. Using complex algorithms for parameter identification and optimization, APSSs can provide good damping over a wide range of system operating conditions [128-131]. H_∞ control has also been successfully applied to off-line design of AVRs [132] as well as PSSs [133-134]. Both APSS and H_∞ control have demonstrated that it is possible to achieve much better performance than with CPSSs. However, most modern control techniques require extensive mathematical calculations, which implies the need for high-speed processors, and high implementation costs.

Recently, Artificial Intelligence (AI) control techniques have been applied as alternatives for both the conventional control and modern control techniques. Table 8.1 gives a snapshot on the state-of-the-art and an overview on the AI applications in synchronous-machine control. For more details, the reader is referred to [166] as a comprehensive review. Four AI techniques have been used in the literature so far. Expert systems are used more often as a supervisory controller than as an automatic controller. They are useful in assisting the human operator but can not replace him/her completely. Neural networks have been successfully applied; however, they have limitations in handling qualitative knowledge, and it is very difficult to come to a reasonable interpretation of the overall structure of the network in terms of humanly understandable concepts. Fuzzy logic controllers interpret the expert knowledge into a form of If-Then rules, but they lack a systematic way to find their parameters.

Fuzzy-neural-networks based controllers utilize the strengths of both fuzzy logic and neural networks. Also, genetic-based fuzzy approaches present powerful optimization and synthesis tools for a controller design. However, still not enough work has been done in the application of these approaches in synchronous-machine control. Therefore, the research using the latter approaches is encouraged due to their power in handling both quantitative and qualitative knowledge.

Many AI-based synchronous-machine controllers have been successfully implemented and tested in labs [149, 150, 153, 155]. However, not many controllers have been implemented on-line in industry and/or power-stations. The reasons for that could be the state of the AI techniques as a new technology, and the usual conservativeness of the industrial utilities for upgrading a running system.

Table 8.1 An overview of AI applications to synchronous machines control.

Application	References	Approach	Comments
Stability Assessment	135, 136	Expert System	Determines the exact cause of the instability, and helps the operator to maintain the system stability.
PSS	137	ES (rule-based)	Uses heuristic if-then rules.
PSS	138-144	Neural Network	Different back-propagation techniques.
PSS	145	Neural Network	Redial basis function for lead-lag controller.
PSS	146	Neural Network	Nonlinear power flow dynamics.
PSS	147-154	Fuzzy Logic	Heuristic rules, Lab tests.
AVR	155	Fuzzy Logic	Heuristic rules, Lab tests.
PSS stability	156	Fuzzy Logic	Shows robustness of FL PSSs.
PSS	157	FL + NN	Fuzzy control+Neuro-prediction routine.
PSS	158	Fuzzy Logic	Self-Organizing, ARMA model.
PSS	159	Fuzzy Logic	Modification of terminal feedback voltage.
PSS + AVR	160	Fuzzy Logic	Two control loops.
PSS	161	LQR + FNN	Complicated Design.
PSS	162	Neuro-Fuzzy	Self-organizing, heuristic performance index.
PSS	163	Neuro-Fuzzy	Large training set, emulates an APSS.
AVR	87	Neuro-Fuzzy	Off-line 3-phase self-learning approach.
PSS	164, 165	Fuzzy-Genetic	Optimized controller, many assumptions.

In this chapter, the hybrid learning scheme proposed in Chapter 3 is used to design three different control schemes for synchronous machines. The first controller is a Neuro-Fuzzy Automatic Voltage Regulator (NF AVR) for a synchronous generator to improve its voltage stability. The second controller is a Neuro-Fuzzy Power System Stabilizer (NF PSS) for a single-machine infinite-bus system. The third one is a neuro-fuzzy power system stabilizer for a synchronous machine in a multi-machine power-system environment to suppress multi-mode oscillations.

8.3 NF AVR for a synchronous generator

Figure 8.2 shows the block diagram of the excitation-control system in the classical feedback control form. The proposed Neuro-Fuzzy Controller (NFC) is used as an Automatic Voltage Regulator (AVR) for a synchronous generator [87]. We refer later to this controller as the Neuro-Fuzzy Automatic Voltage Regulator (NF AVR). The inputs to this NF AVR are the error and the rate of change of the generator terminal voltage (del_volt), as shown in Figure 8.2.

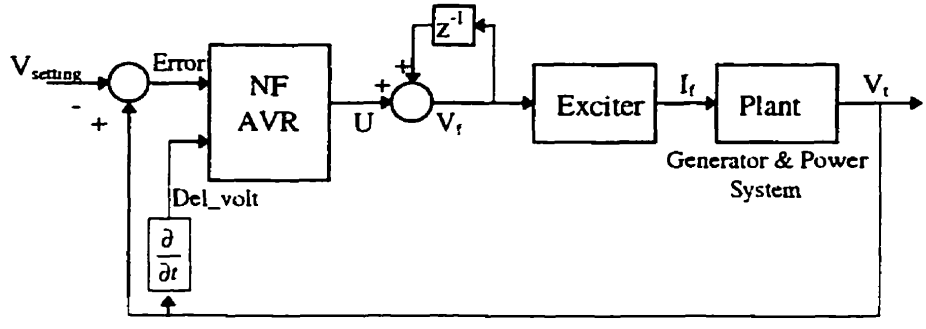


Figure 8.2 Excitation control system using the NF AVR.

Three scaling factors G_e, G_d, and G_c are used within the NF AVR to adjust the input/output values of the controller into proper ranges [167]. With the help of these factors, we can say that the NF AVR consists of a normalized NF AVR and three scaling factors, as shown in Figure 8.3. In the normalized NF AVR, the centers of the input/output membership functions of the controller lay in the range of [-1, 1].

8.3.1 Pre-trained NF AVR

The NF AVR is initially designed from the author experience and knowledge of the control objectives. In this design, the centers of the input/output variables are distributed regularly on the specified range of the normalized NF AVR [-1, 1]. Figure 8.4 shows the distribution of the bell-shaped membership functions of the three variables of the normalized NF AVR. The variances (widths) of all the functions are equal, and each has the value of '1/6'.

The scaling factors of the initial design of the NF AVR are selected to be 'G_e = 1/60', 'G_d = 1/10', and 'G_c = 1/16'. These settings are based on author's experience and on trial and error procedures. The control rules of this initial design are shown in the Fuzzy Associative Memory (FAM) matrix shown in Table 8.2. Logic, intuition, experience and knowledge of control objectives have been used to form these rules.

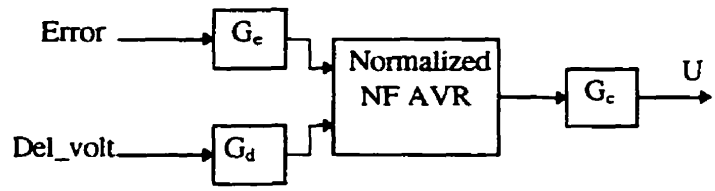


Figure 8.3 Topology of the NF AVR.

Table 8.2 The FAM matrix with initial rules.

	error						
del_volt	NL	NM	NS	ZE	PS	PM	PL
NL	PL	PL	PL	PL	PM	PS	ZE
NM	PL	PL	PM	PM	PS	ZE	NS
NS	PL	PM	PS	PS	NS	NM	NL
ZE	PL	PM	PS	ZE	NS	NM	NL
PS	PL	PM	PS	NS	NS	NM	NL
PM	PM	ZE	NS	NM	NM	NL	NL
PL	ZE	NS	NM	NL	NL	NL	NL

All entries correspond to out_vlot

The intuitive version of the NF AVR is applied to control the terminal voltage of the synchronous generator. The reference-setting of the terminal voltage is 220V. Figure 8.5 shows the response of the synchronous generator during severe load-resistance changes; the disturbances caused by the load-resistance are exhibited in Figure 8.6. The simulation sampling time is 0.15 seconds. The simulation study in this section is based on a simple linear model of the synchronous generator that assumes a constant speed during load disturbances. Figure 8.5 contains 10,600 samples that will be used as training patterns to design a learned NF AVR. Each pattern consists of two elements $[V_f(k), V_t(k)]$. These training patterns include a lot of information about the dynamical behavior and input/output properties of the synchronous generator.

8.3.2 First Learning Phase of The NF AVR

The self-organizing feature map (SOM) algorithm is used to find the initial centers and widths of the learned version of the NF AVR, as described before in Section 4.2. The training examples shown in Figure 8.4 are used as the input data to this algorithm. The resultant membership functions, after this unsupervised learning process, are shown in Figure 8.7. According to the results of the self-organized map algorithm, the scaling factors of the NF AVR are adapted to be 'G_e = 1/19.5', 'G_d = 1', 'G_c = 1/58.4'. The NF AVR, after the first learning phase, is used to control the synchronous generator during the occurrence of the disturbances shown in Figure 8.6. The response of the

generator is shown in Figure 8.8. It is clear that the settling time of the terminal voltage is improved due to the SOM.

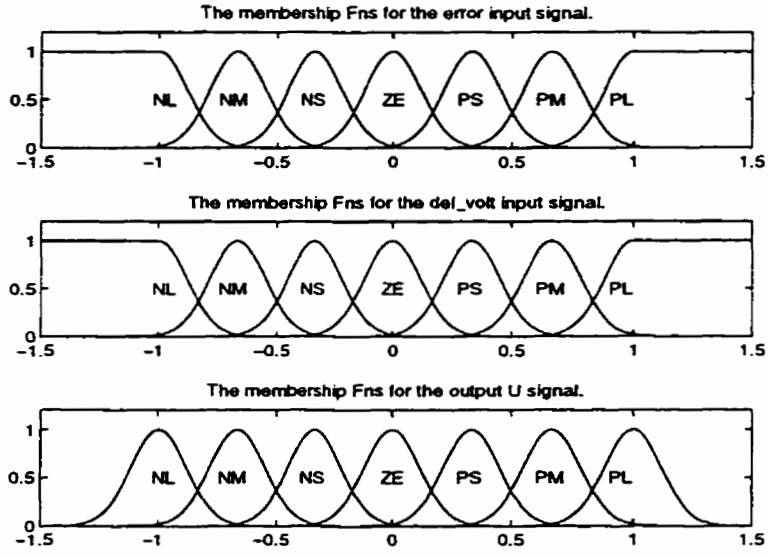


Figure 8.4 The membership functions of the intuitive NF AVR.

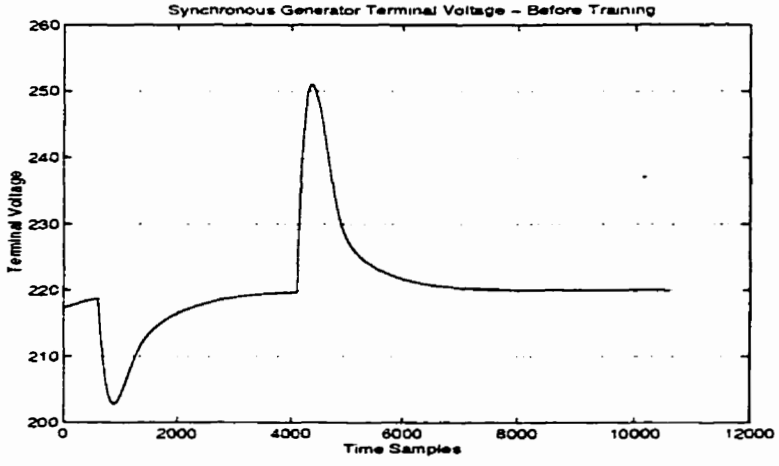


Figure 8.5 Synchronous machine response-before training.

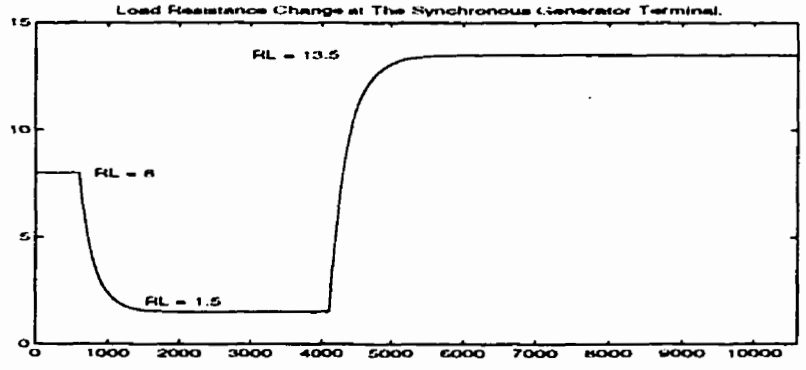


Figure 8.6 Resistive-load disturbances.

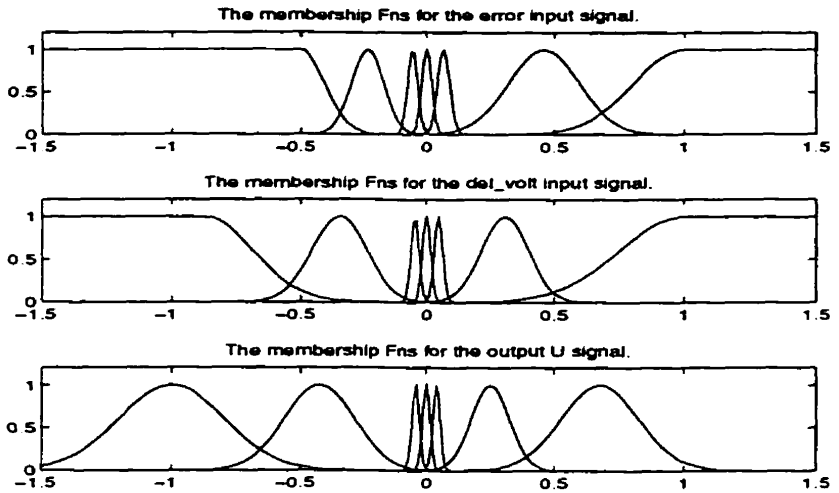


Figure 8.7 NF AVR membership functions after SOM.

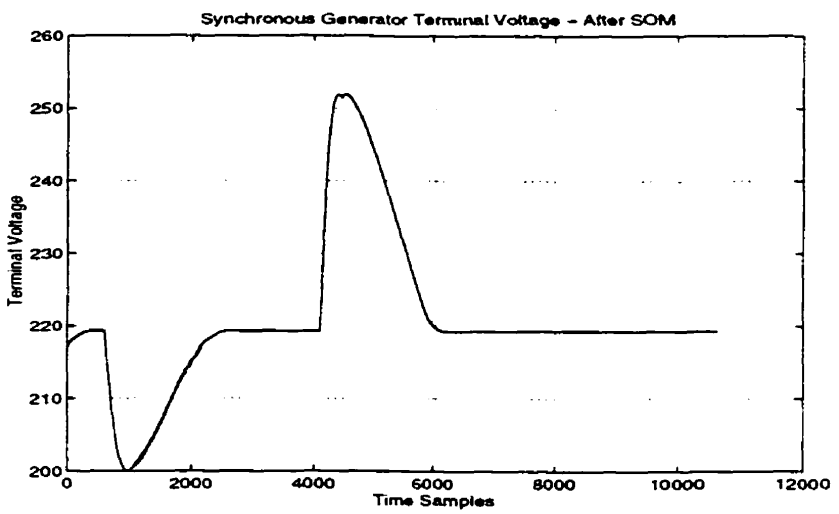


Figure 8.8 The generator response after SOM.

8.3.3 Second Learning Phase of The NF AVR

The Minimum Distance Algorithm (MDA) proposed by the author, and described in Section 5.2.2, is used to find the rules of the NF AVR. In Table 8.3, the rules that their linguistic values written in **bold-*italic*** face, are found by the MDA algorithm, and then the rest of the rules are found using straight forward logic by smoothing and extrapolating the generated rules. Logic are mainly used in some rule-space regions that do not have enough training data to produce consistent fuzzy rules.

After the new rules have been found, the NF AVR after the second learning phase, is used to control the synchronous generator during the disturbances shown in Figure 8.6; the response is exhibited in Figure 8.9. It is obvious that the settling-times of the terminal voltage are better than before, while, the over/under-shoots have not

significantly changed.

	error						
del_volt	NL	NM	NS	ZE	PS	PM	PL
NL	PL	PL	PL	***PL***	NL	NM	***NL***
NM	PL	***PL***	***PL***	***PM***	***NM***	***NM***	NL
NS	PL	***PL***	***PL***	***PS***	***NS***	***NM***	NL
ZE	PL	PM	PS	***ZE***	NS	NM	NL
PS	PL	***PM***	***PS***	***NS***	***NL***	***NL***	NL
PM	PL	***PM***	***PM***	***NM***	***NL***	***NL***	NL
PL	***PL***	PM	PM	***NL***	NL	NL	NL

All entries correspond to out_vlot

Table 8.3 The FAM matrix with new rules.

8.3.4 Third Learning Phase of The NF AVR

In this learning phase, the error back-propagation algorithm (described in Section 6.2) is used to optimize the membership functions of the NF AVR. The fuzzy-neural network is trained to learn the inverse dynamics of the synchronous generator. Here, the learning rate (η) is 0.1 and the accepted average RMS error is ≤ 0.05. Figure 8.10 shows the optimized membership functions of the NF AVR after this learning phase.

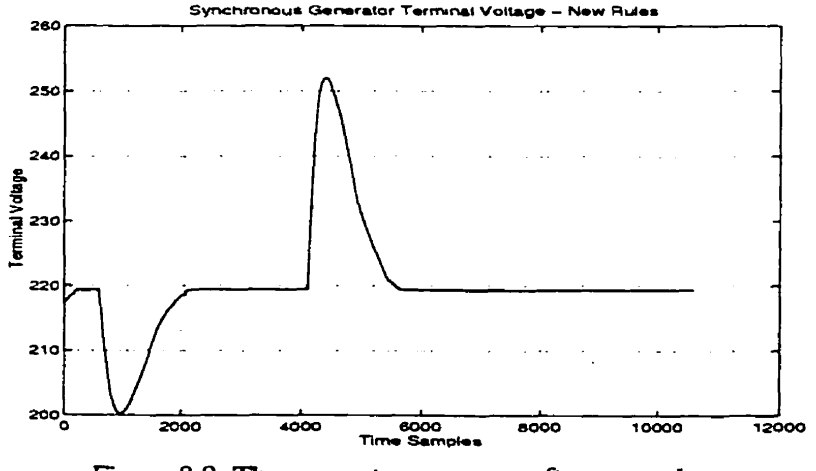


Figure 8.9 The generator response after new rules.

After the NF AVR is already learned from the input/output properties of the synchronous generator, it has been applied to control the generator while the disturbances shown in Figure 8.6 occur. The response of the generator is shown in

Figure 8.11. It is clear that the settling times of the terminal voltage are much better than those before learning (around 45% decrease). Also, the over/under-shoots are decreased by about 10%.

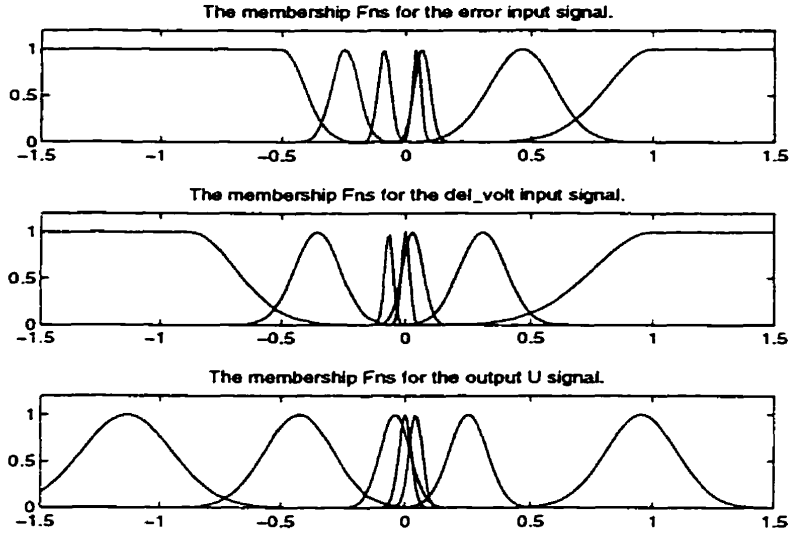


Figure 8.10 The optimized membership functions.

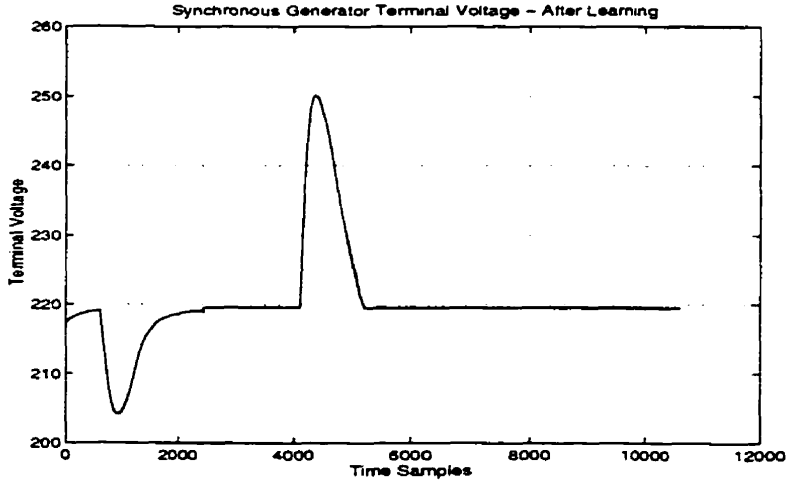


Figure 8.11 The response after the learning process has been completed.

Extensive simulation studies of the synchronous generator have been carried out to prove the effectiveness of the NF AVR. The studies show that the synchronous-generator response has improved during the learning phases of the NF AVR. Simulation also shows that the generator response, after the learning phases have been completed, is much better than the pre-learned response; in terms of settling times, over/under-shoots, and rising/falling times.

8.4 NF PSS for a synchronous machine

This study is based on a detailed model of a single-machine (5^{th} order model [112]) connected to a constant voltage bus (infinite bus) through a transformer unit and parallel transmission lines, as shown in Figure 8.12. For comparison purposes, both the NF PSS and CPSS are included in the system.

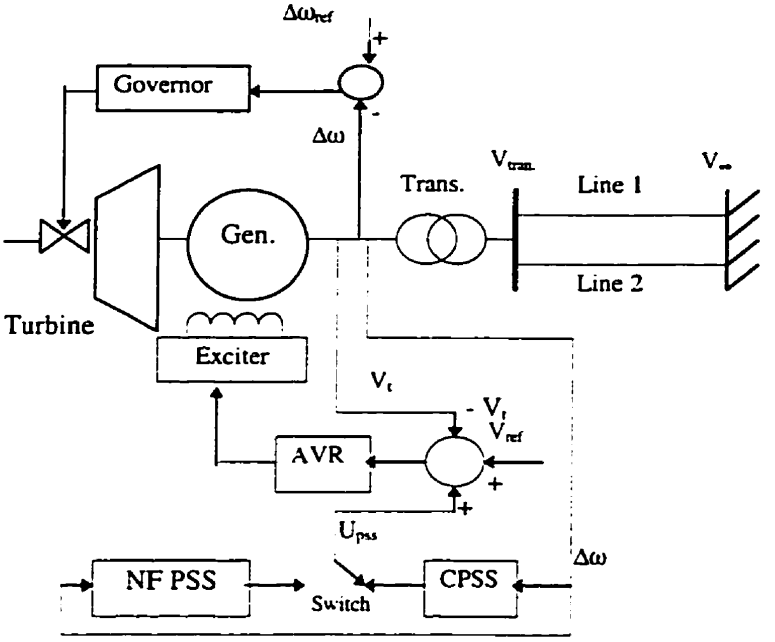


Figure 8.12 Power-system model configuration.

The generator is operating at a terminal power of 0.9 pu and 0.9 power factor lag. Under these conditions, the CPSS is designed and carefully tuned for the best performance, i.e. the overshoot and settling time are minimized. The parameters of the CPSS are kept unchanged for all the tests described in this paper. The output of the CPSS is limited in the range [-0.1pu, 0.1pu] and its transfer function is given as:

$$U_{PSS}(s) = 0.1\frac{2.5s(1+0.1s)^2}{(1+2.5s)(1+0.03s)^2}\Delta\omega \qquad (8.1)$$

The inputs to the proposed NF PSS are the speed deviation ($\Delta\omega$) and the rate of speed-change ($\Delta\varpi$). The NF PSS consists of a normalized NF PSS and three scaling factors K_{ω}, $K_{d\omega}$, and K_u, as shown in Figure 8.13. In the normalized NF PSS, the centers of the membership functions of all the controller variables lay in the range [-1, 1].

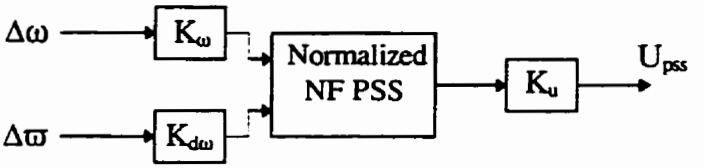


Figure 8.13 Topology of the NF PSS.

The training data for the NF PSS is collected from the measurements of the speed-deviation of the synchronous machine while undergoing oscillations due to a 3-phase short circuit on Line 2 at the transformer bus. The short circuit is cleared after 0.17 sec by opening the second transmission line. This fault type is selected to cover the widest possible range of oscillations. The output of the CPSS has been disconnected by opening the switch during the fault and included in the training data. A total of 400 samples are collected and shown in Figure 8.14.

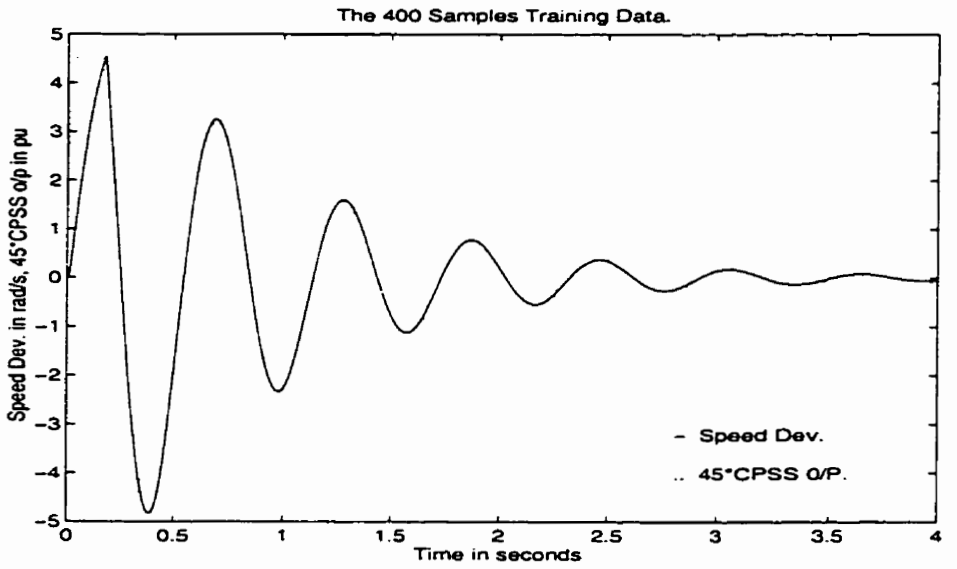


Figure 8.14 The NF PSS training data.

The partitioning of the normalized NF PSS is chosen, by the author's experience in fuzzy-controllers design, to be seven for each input/output variable ($n_1=n_2=n_3=7$). The SOM algorithm (Section 4.2) is used to determine the initial centers and widths (with r=2.0) of the 21 membership functions of the input/output variables from the data shown in Figure 8.14. The centers of NL and PL membership functions of the NF PSS output are fixed at -1 and 1, respectively, and the K_u scaling factor is set to be 0.1, in order to keep the NF PSS output in the range [-0.1pu, 0.1pu]. The K_ω and $K_{d\omega}$ scaling factors are determined from the SOM as 0.3925 and 0.03 respectively.

The MMFA algorithm (Section 5.2.3) is then used to find the linguistic-fuzzy rules. The author's experience in PSS's design is also employed to find out and/or check the rules, especially

in the regions where low density of training data exists. The extracted 49 rules are shown in Table 8.4.

Table 8.4 The fuzzy associative memory matrix with the rules.

	Δω						
Δϖ	NL	NM	NS	ZE	PS	PM	PL
NL	*NL*	*NL*	*NL*	*NL*	*NL*	*NL*	*NL*
NM	*NL*	*NL*	*NL*	*NL*	*NS*	*NS*	*NS*
NS	*NL*	*NM*	*NM*	*NS*	*ZE*	*PM*	*PM*
ZE	*NM*	*NM*	*NS*	*ZE*	*PS*	*PM*	*PM*
PS	*NM*	*NS*	*ZE*	*PS*	*PM*	*PM*	*PL*
PM	*ZE*	*PS*	*PM*	*PM*	*PL*	*PL*	*PL*
PL	*PS*	*PM*	*PL*	*PL*	*PL*	*PL*	*PL*

All entries correspond to U_{PSS}

The MRD-GA algorithm (Section 6.3) is then applied to optimize the parameters of the NF PSS. The algorithm parameters are set as follows: population size = 50, probability of crossover = 0.9, probability of mutation = 0.05, chromosome-length = 37, $\delta_m=\delta_\sigma=0.0001$, $\theta_m=\theta_\sigma=0.99$, and $\tau=20$. The centers of the ZE membership functions of the three NF PSS variables are fixed at 0.0, and the centers of NL and PL membership functions of the output are fixed at -1 and 1, respectively. Accordingly, there are 16 centers and 21 widths to be tuned by the MRD-GA.

The objective of the MRD-GA is to

$$Min\{\int \Delta\omega(t)^2 \; dt\} \quad subject\ to \quad f(0,0)=0 \tag{8.2}$$

where $U_{PSS} = f(\Delta\omega, \Delta\varpi)$ is the fuzzy controller mapping. This constraint ensures zero steady-state error of the controller; for the same reason, the ZE membership functions are also centered at 0.0. The MRD-GA algorithm stopped after 2000 generations and the objective function decrease rate is shown in Figure 8.15. The response of the synchronous machine after the 3-phase fault, with the CPSS and the trained NF PSS, is also shown in Figure 8.16.

A number of studies have been performed to investigate the effectiveness of the proposed NF PSS and to compare its results with those of the CPSS.

- **Test 1:** The generator is operating at 0.9pu power and 0.9 pf lag, and a 20% step increase in the input mechanical power (P_m) is applied and then removed after 4.5 seconds. System response, using the CPSS and NF PSS, is shown in Figure 8.17.

- **Test 2:** The generator is operating at 0.2pu power and 0.9 pf lead, and a 50% step increase in P_m is applied and then removed 4.5 seconds later. The responses are shown in Figure 8.18.

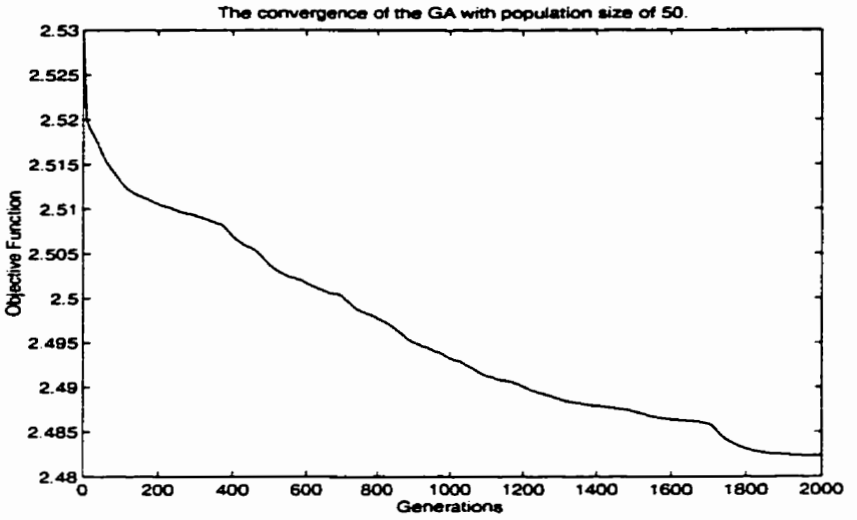


Figure 8.15 The MRD-GA convergence rate.

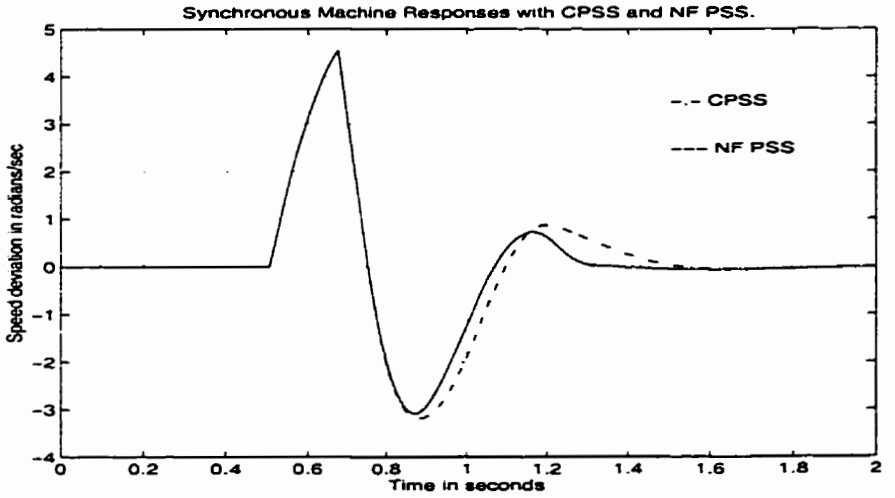


Figure 8.16 Response to 3-ph fault at power 0.9pu, 0.9 pf lag.

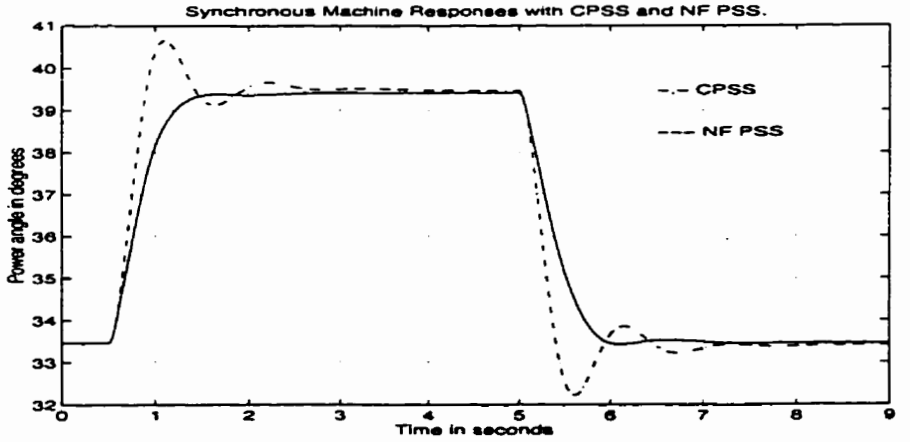


Figure 8.17 Response to 20% increase in P_m (0.9pu, 0.9 pf lag).

- **Test 3:** The generator is operating at 0.9pu power and 0.9 pf lag, and a 3% step increase in the reference voltage (V_{ref}) is applied and then removed after 4.5 seconds. The system response, using the CPSS and NF PSS, is shown in Figure 8.19.

- **Test 4:** The generator is operating at 0.9pu power and 0.9 pf lag, and Line 1 is opened and then closed 4.5 seconds later. The system response is shown in Figure 8.20.

- **Test 5:** To study the sensitivity of both the CPSS and NF PSS to the imprecision or even vagueness in system parameters, the responses of the generator with different values of inertia-constants (H=4, 10, and 16) are shown in Figures 8.21 and 8.22, respectively. The generator is operating at 0.4pu power and 0.9 pf lag, while a 80% step increase in P_m is applied and then removed after 4.5 seconds. Of course, it is not realistic that the inertia-constant of a synchronous generator changes after installation. However, the main purpose of this test is to show the capability of both controllers to cope with the change in the system's configuration.

- **Test 6:** The generator is operating at 0.9pu power and 0.9 pf lag, and a 3-phase short circuit occurred at the middle of Line 1 and then cleared after 0.17 sec by opening the line. The system response is shown in Figure 8.23.

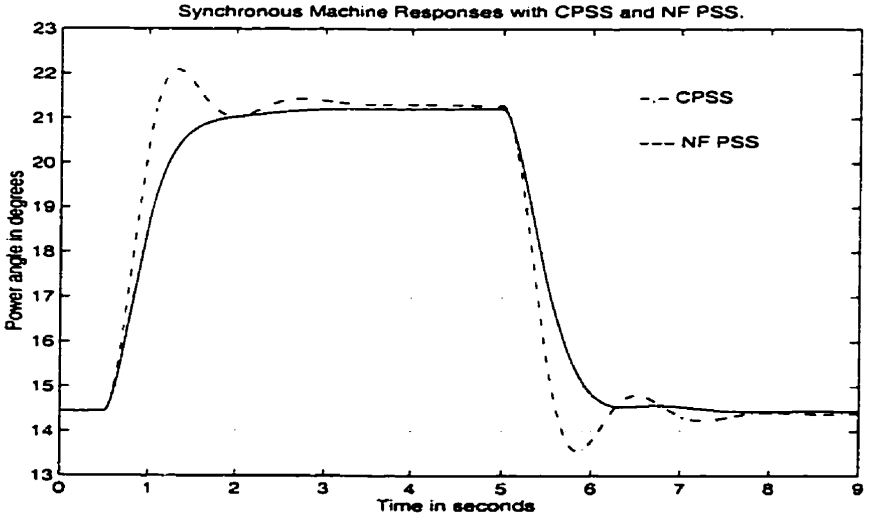


Figure 8.18 Response to 50% increase in P_m (0.2pu, 0.9 pf lead).

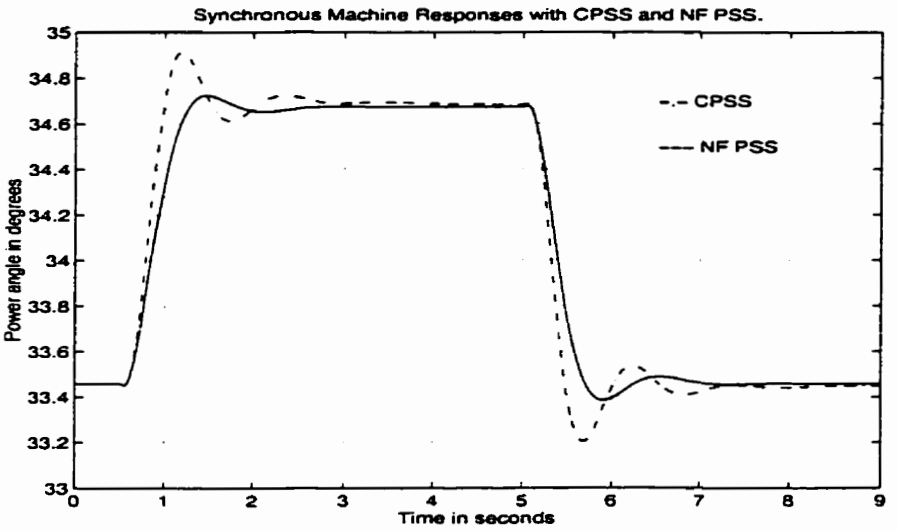


Figure 8.19 Response to 3% increase in V_{ref} (0.9pu, 0.9 pf lag).

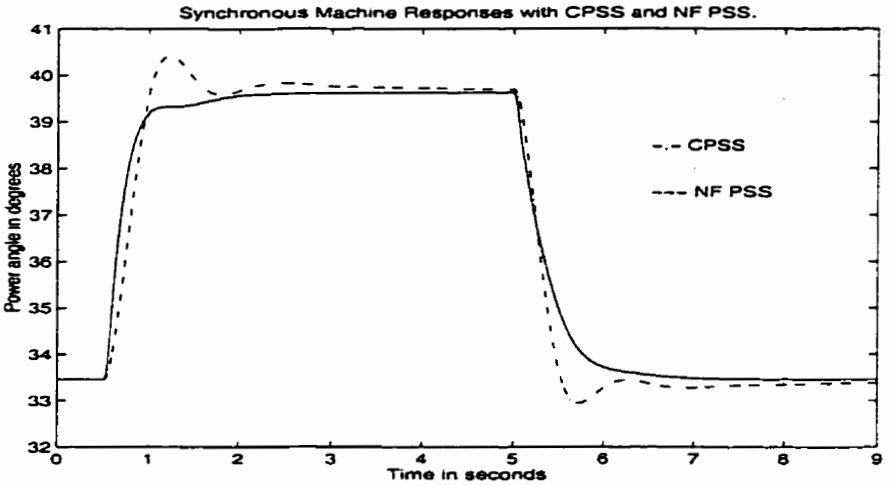


Figure 8.20 Response to Line 1 switching (0.9pu, 0.9pf lag).

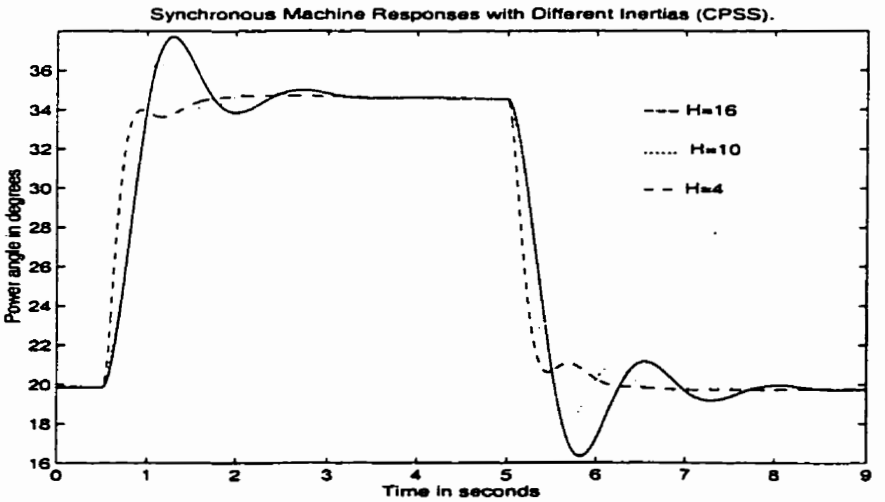


Figure 8.21 Response to different inertia constants (CPSS).

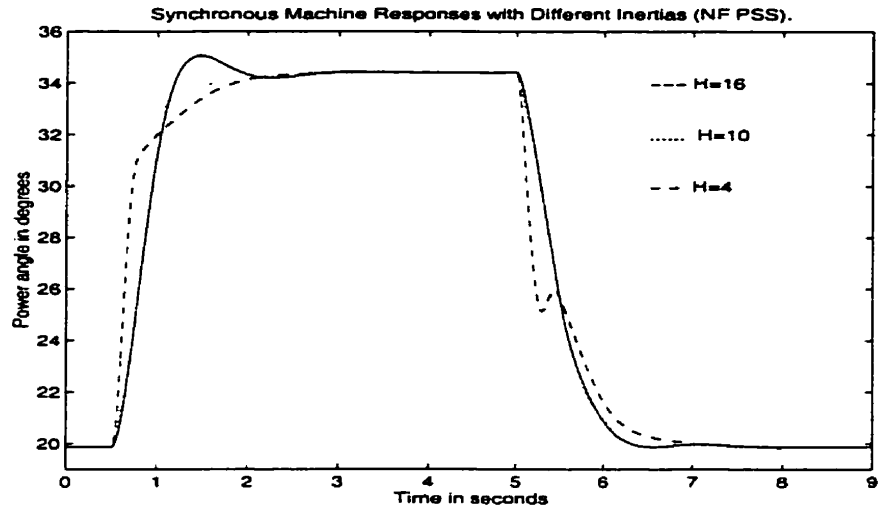


Figure 8.22 Response to different inertia constants (NF PSS).

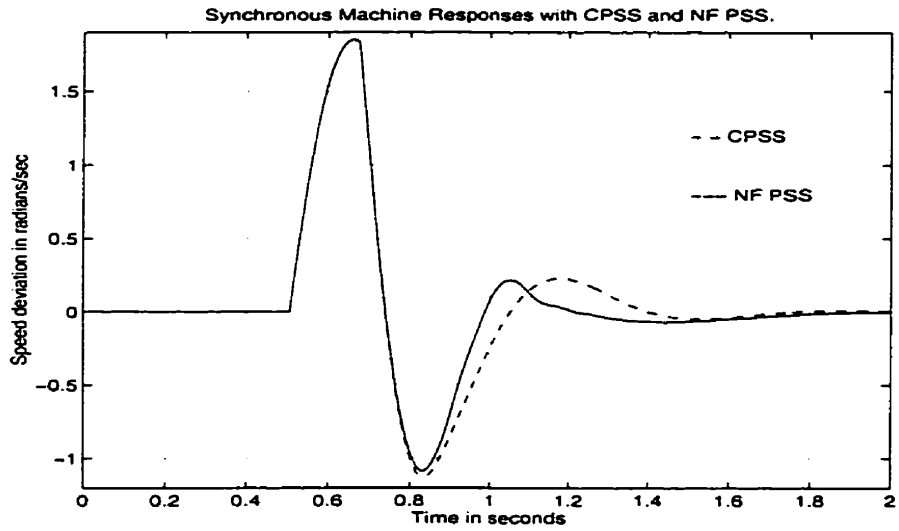


Figure 8.23 Response to 3-phase fault at the middle of Line 1.

Test results for various conditions show that the proposed NF PSS has better performance than the CPSS. It provides good damping over a wide operating range and significantly improves the transient and dynamic performance of the synchronous machine. The NF PSS shows also less sensitivity to the change of the system parameters than the CPSS as illustrated by Figures 8.21 and 8.22. The approximate reasoning feature of the NF PSS makes it capable of coping with different power-system configurations with satisfactory performance, without the need to be re-tuned. A clear advantage of the presented design method is the use of a compact-size training data (400 samples); while in *Farag et al.* [87] and *Hariri et al.* [163] 10,600 and 18,000 samples are used, respectively. Also, in the third phase an objective function, which can be tailored according to the designer needs, is used. The above features make the design method systematic and more convenient.

8.5 NF PSS for multi-machine power-system environment

To evaluate the effectiveness of the proposed NF PSS, a transient stability simulation study is performed on the nine-bus three-machine power system shown in Figure 8.24. Each machine is represented by a fourth-order two-axis nonlinear model. Details of the system's data can be found in [110, ch. 3].

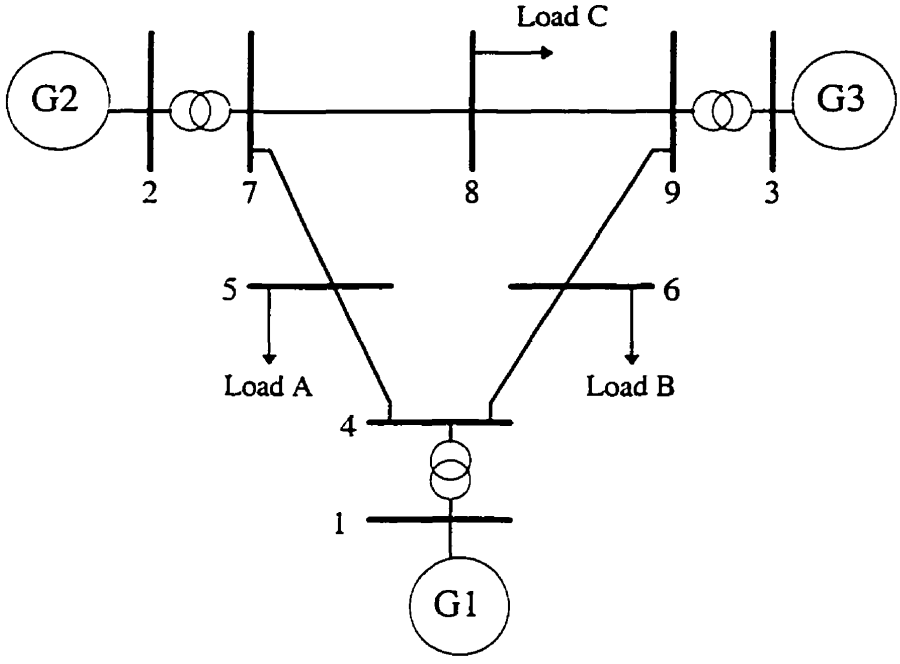


Figure 8.24 Three-machine nine-bus power system model.

The details of the operating point of the power system are given in Table 8.5. Under these operating conditions of Table 8.5, three CPSSs are designed, one for each generator. The parameters of the CPSS are kept unchanged for all the tests performed and reported in this study. The outputs of the CPSSs are limited in the range [-0.1pu, 0.1pu] and their transfer functions are identical and given as [110]:

$$U_{PSS}(s) = \frac{10s(1+0.568s)^2}{(1+10s)(1+0.0227s)^2}\Delta\omega \qquad (8.3)$$

Table 8.5 Loading Conditions.

	P (pu)	Q (pu)	Terminal Volt.
G1	0.716	0.027	1.040∠0.0°
G2	1.630	0.0067	1.025∠9.3°
G3	0.850	-0.109	1.025∠4.7°
Load A	0.125	0.050	0.996∠-4.0°
Load B	0.090	0.030	1.013∠-3.7°
Load C	0.100	0.035	1.016∠0.7°

The inputs to the proposed NF PSS are the speed deviation ($\Delta\omega$) and the rate of the speed-change ($\Delta\varpi$). The NF PSS consists of a normalized NF PSS and three scaling factors K_{ω}, $K_{d\omega}$, and K_u, as shown in Figure 8.25. In the normalized NF PSS, the centers of the membership functions of all the controller variables lie in the range [-1, 1].

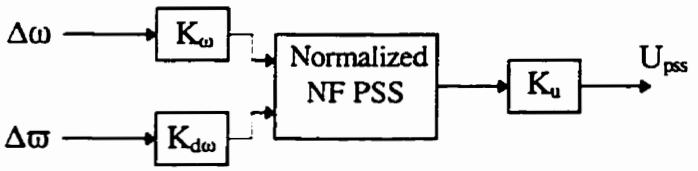


Figure 8.25 Topology of the NF PSS.

An optimized NF PSS is designed for generator (G2). The training data for this NF PSS is collected from the measurements of the speed-deviation of the generator which undergoes oscillations due to a 3-phase short circuit on Line 5-7, adjacent to Bus 7. The short circuit is cleared after 8 cycles by successful reclosing of Line 5-7. This fault type is selected to cover the widest possible range of system's oscillations. A total of 600 data samples are collected; they are plotted as in Figure 8.26.

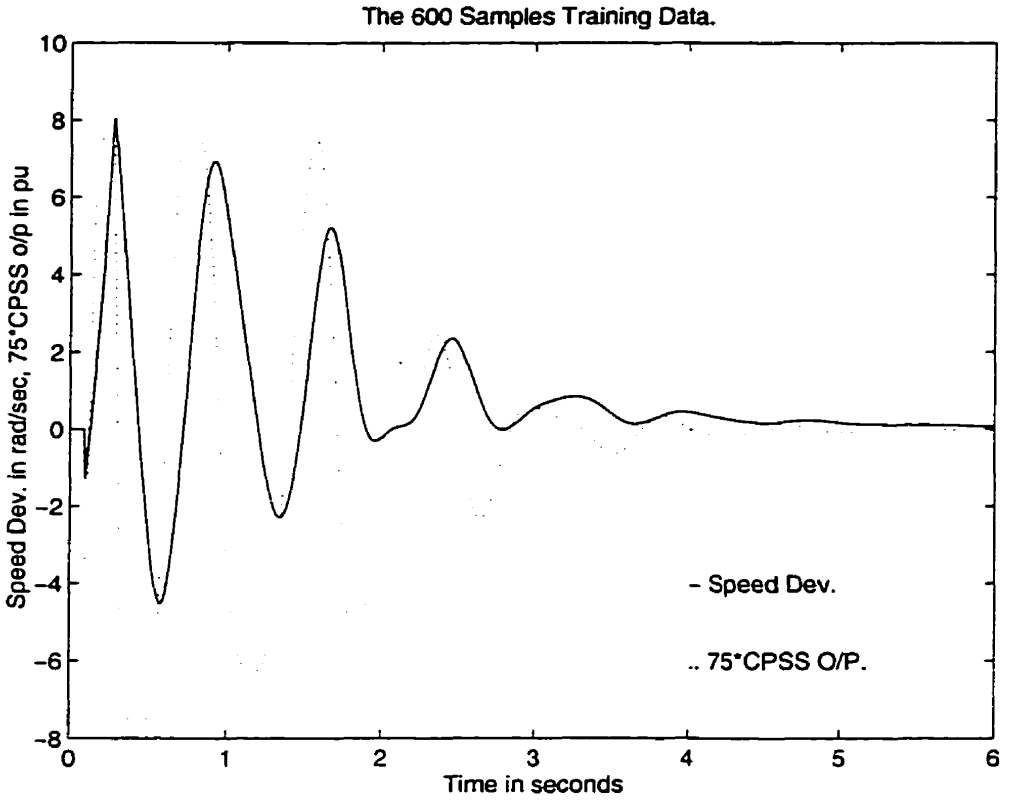


Figure 8.26 The NF PSS training data.

The FCM algorithm (Section 4.3) is used to determine the most appropriate fuzzy partitions for the input/output variables and the initial centers and widths of their membership functions from the data shown in Figure 8.26. Table 8.6 shows the results of the clustering assessment using the *S3* clustering validity measure (CVM) mentioned in Section 4.4 (Equation 4.17). For each NF PSS variable, the shaded area contains the lowest value for the *S3* CVM, and the associated number of clusters (c) is considered the most appropriate fuzzy partition. Accordingly, the best partitioning of the normalized NF PSS is found to be seven for the $\Delta\varpi$ input (n_2=7) and nine for both the $\Delta\omega$ input and U_{pss} output (n_1=n_3=9). However, in this design we will use (n_1=7) to decrease the number of rules from 63 to 49 (more compact-size knowledge base) and to produce a simpler fuzzy controller.

The centers of the NL and PL membership functions of the NF PSS output are fixed to -1 and 1, respectively; and the K_u scaling factor is set to be 0.1 in order to keep the NF PSS output in the range [-0.1pu, 0.1pu]. The K_ω and $K_{d\omega}$ scaling factors are determined from the FCM algorithm as '0.1478' and '0.0195', respectively.

Table 8.6 Clustering assessment by *S3* CVM.

c	$\Delta\omega$	$\Delta\varpi$	U_{pss}
2	-7.4261	-1.6797	-0.2684
3	-9.8097	-138.0168	-184.097
4	-15.5272	-136.7000	-192.228
5	-16.4018	-165.3186	-196.943
6	-16.3481	-168.4599	-193.490
7	-16.3261	-171.7263	-197.499
8	-17.4982	-169.0656	-200.511
9	-17.6664	-166.9547	-200.762
10	-17.3814	-165.0680	-200.193
11	-17.5540	-160.9201	-200.569

The SGA algorithm (Section 5.2.4) is then used to find the linguistic-fuzzy rules. The algorithm parameters are set as follows: population size = 80, probability of crossover = 0.9, probability of mutation = 0.15, chromosome-length = 49. The computation time elapsed to perform this learning phase is about 20 minutes on a Pentium 166MHz. The author's experience in PSS's design is also employed to find out and/or check the rules, especially in the portions where low density of training data exists. The extracted 49 rules are shown in Table 8.7; the convergence curve of the SGA is exhibited in Figure 8.27.

Table 8.7 The fuzzy associative memory matrix with the rules.

	$\Delta\omega$						
$\Delta\varpi$	NL	NM	NS	ZE	PS	PM	PL
NL	*NL*	*NL*	*NL*	*NL*	*NL*	*NL*	*NM*
NM	*NL*	*NM*	*NM*	*NM*	*NM*	*ZE*	*PS*
NS	*NM*	*NS*	*NS*	*NVS*	*ZE*	*PVS*	*PS*
ZE	*NS*	*NS*	*NS*	*ZE*	*PVS*	*PS*	*PS*
PS	*NVS*	*NVS*	*PS*	*PS*	*PS*	*PM*	*PM*
PM	*PS*	*PS*	*PS*	*PM*	*PM*	*PL*	*PL*
PL	*PS*	*PM*	*PM*	*PM*	*PL*	*PL*	*PL*

All entries correspond to U_{PSS}

Figure 8.27 The SGA convergence rate.

The MRD-GA (Section 6.3) is then applied to optimize the parameters of the NF PSS. The algorithm parameters are set as follows: population size = 100, probability of crossover = 0.9, probability of mutation = 0.05, chromosome-length = 41, $\delta_m=\delta_\sigma=0.0001$, $\theta_m=\theta_\sigma=0.99$, and $\tau=20$. The centers of the ZE membership functions of the three NF PSS variables are fixed at 0.0, and the centers of NL and PL membership functions of the output are fixed at -1 and 1, respectively. Accordingly, there are 18 centers and 23 widths to be tuned by the MRD-GA.

The objective of the MRD-GA is to

$$Min\{\int \Delta\omega(t)^2 \; dt\} \quad subject\ to \quad f(0,0)=0 \tag{8.4}$$

where $U_{PSS} = f(\Delta\omega, \Delta\varpi)$ is the FC mapping. This constraint ensures zero steady state error of the controller. The MRD-GA algorithm has stopped after 1000 generations. The objective function decrease rate is shown in Figure 8.28. The computation time used to perform this learning phase is about 83 minutes on a Pentium 166 MHz PC.

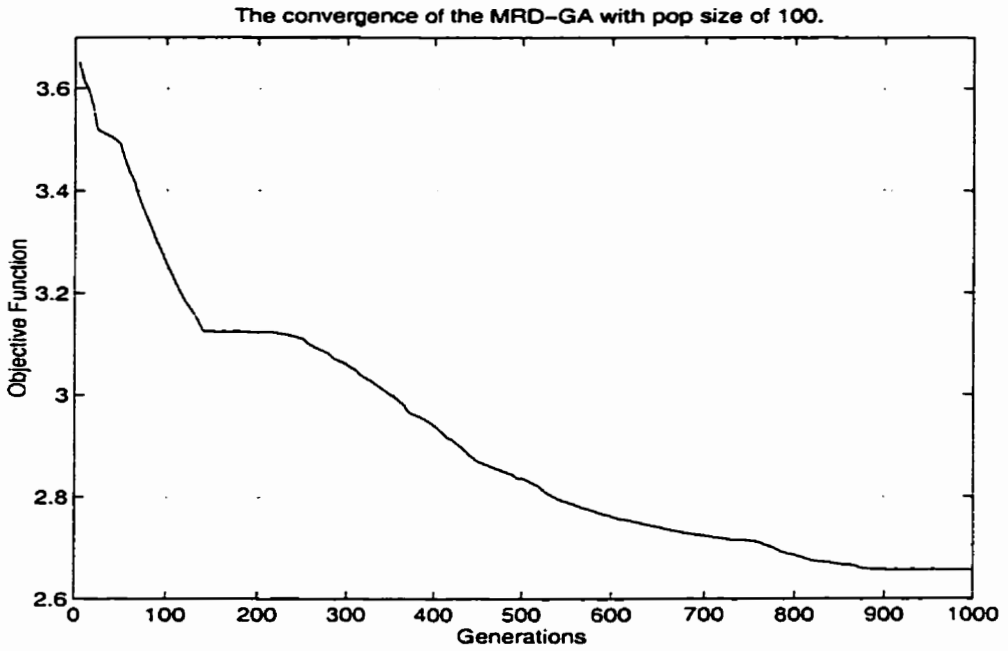


Figure 8.28 The MRD-GA convergence rate.

Several studies have been performed to investigate the effect of the proposed NF PSS and compare the results with those of the CPSS. Some of these studies are presented here.

- **Test 1:** The generators are operating at the operating point indicated in Table 8.5. A 3-phase fault occurs on Line 5-7, near Bus 7. The fault is cleared by opening the line after 8 cycles and then the line is reclosed successfully. In this test, no PSSs are installed on G1 and G3. Figure 8.29 shows the power-system response in two cases; when the CPSS is installed on G2, and when the NF PSS is installed on G2.

- **Test 2:** The power system is operating at the operating point indicated in Table 8.5. A 3-phase fault occurs on Line 5-7, near Bus 7. The fault is cleared by opening the line after 6 cycles without successful reclosing of the line. In this test, a CPSS is installed on both G1 and G3. Figure 8.30 shows the power-system response in two cases; when the CPSS is installed on G2, and when the NF PSS is installed on G2.

- **Test 3:** The generators are operating at the operating point indicated in Table 8.5. Step changes on the input mechanical power of the three generators occur simultaneously as follows: 10% decrease on G1, 25% increase on G2, and 5% decrease on G3. CPSSs are installed on G1 and G2. Figure 8.31 shows the power-system response in two cases; when the CPSS is installed on G2, and when the NF PSS is installed on G2.

- **Test 4:** The generators are operating at the operating point indicated in Table 8.5. Step changes on the input mechanical power of the three generators occur simultaneously as follows: 15%

increase on G1, 20% decrease on G2, and 5% increase on G3. No PSSs are installed on G1 or G3. Figure 8.32 shows the power-system response in two cases; when the CPSS is installed on G2, and when the NF PSS is installed on G2.

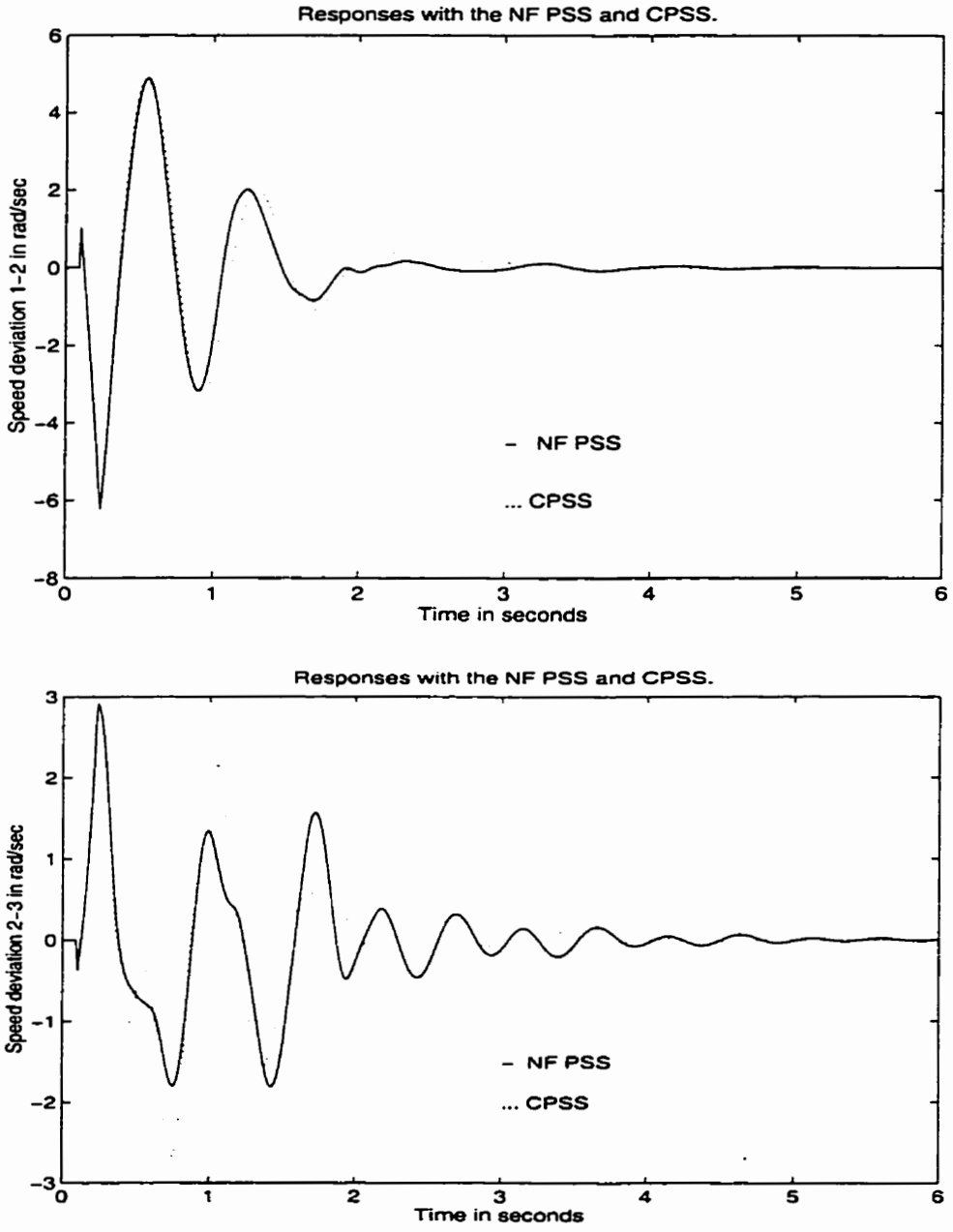


Figure 8.29 Response to a 3-phase fault with successful reclosing.

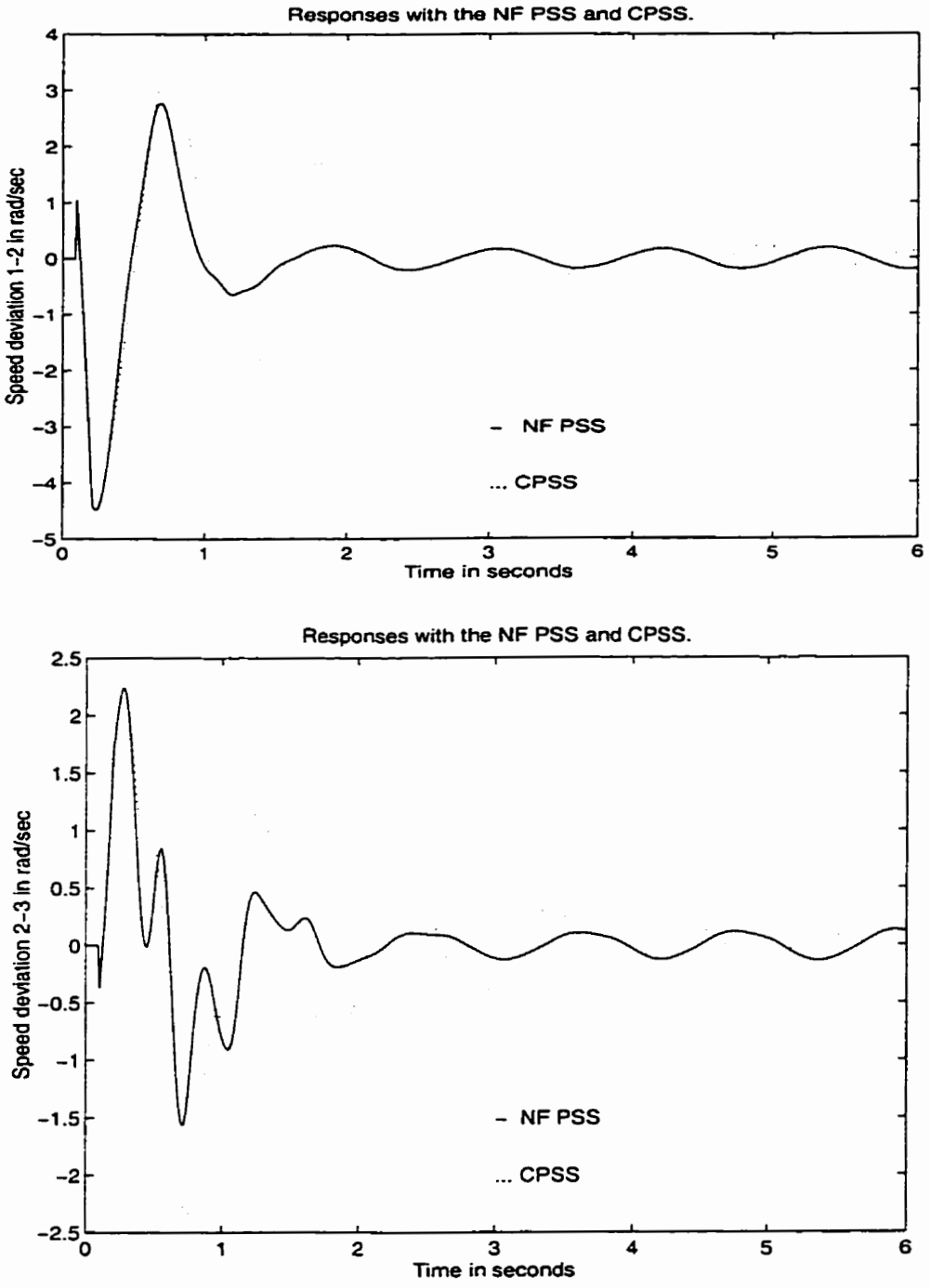


Figure 8.30 Response to a 3-phase fault without successful reclosing.

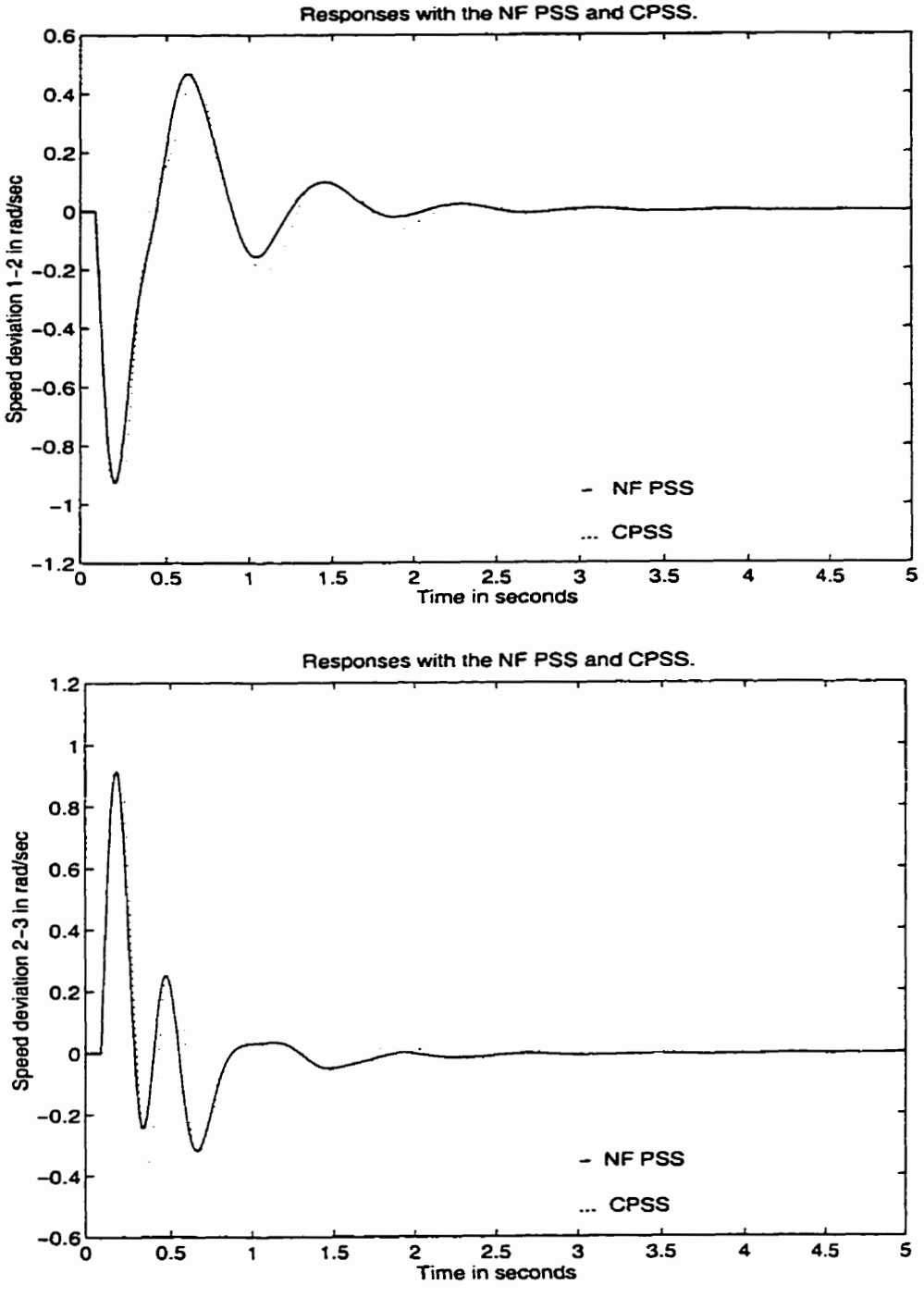


Figure 8.31 Response to step changes in the mechanical power inputs.

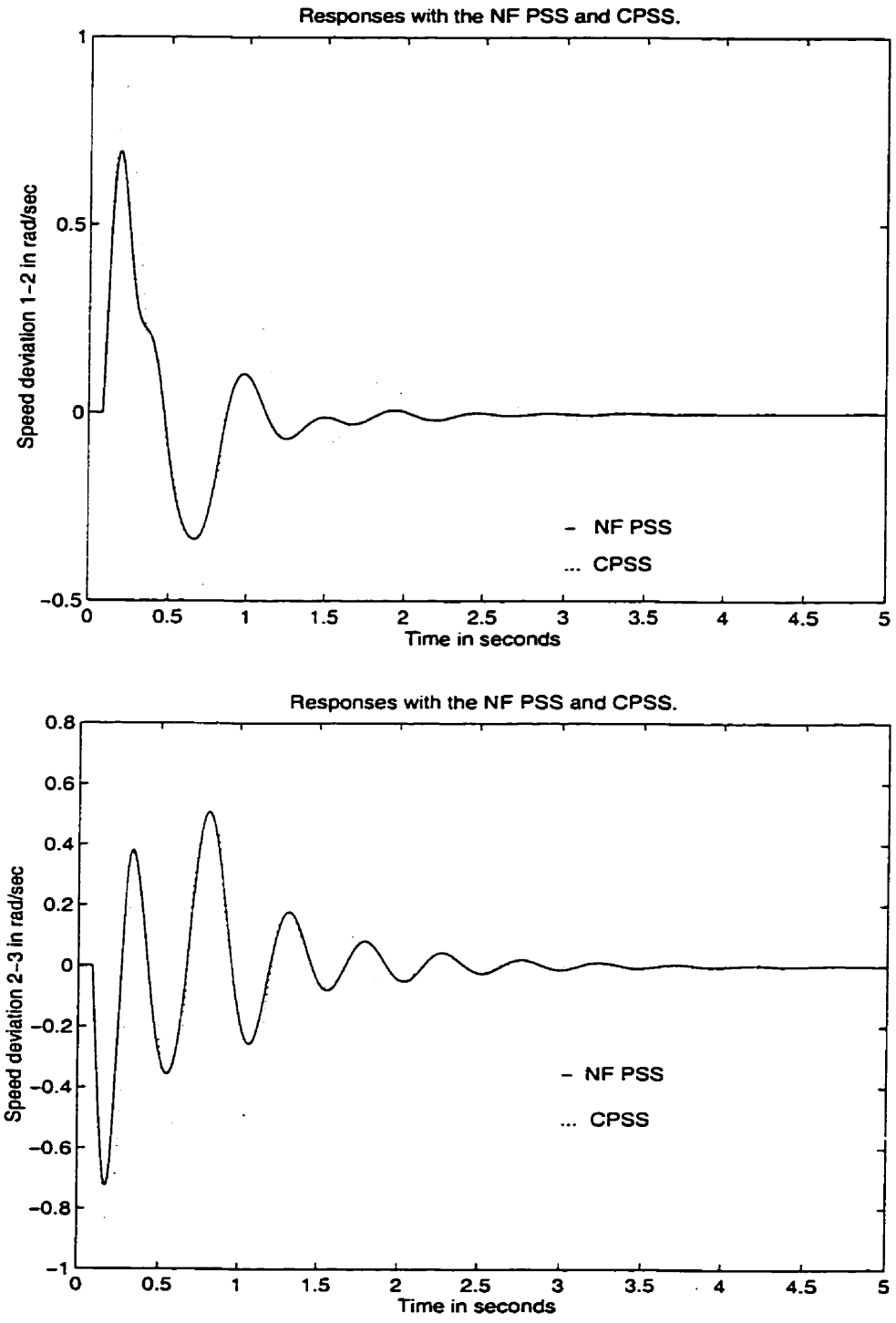


Figure 8.32 Response to step changes in the mechanical power inputs.

In this study, an optimized NF PSS is proposed to improve the transient and dynamic stability of a synchronous generator working in a multi-machine power system environment. Again, only 600 data samples are used here; while in *Farag et al.* [87] and *Hariri et al.* [163] 10,600 and 18,000 samples are used, respectively. Also, in the third learning phase an objective function, which can be tailored according to the designer needs, is used to add to the flexibility of the design method. The above features make the design method systematic and more convenient.

Simulation results for various tests show that the proposed NF PSS has better performance than that of the CPSS. It provides good damping for multi-modes of oscillations under different disturbances and significantly improves the transient and dynamic performance of the overall multi-machine power-system. Moreover, the results show the good coordination and cooperation between CPSSs and the proposed NF PSS while working together in the same power system; as illustrated by Figures 8.29 and 8.32. This feature should encourage power-system engineers to replace conventional power-system stabilizers with neuro-fuzzy power-system stabilizers without the doubt of encountering performance and/or stability problems.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

This thesis introduces a generic framework for synthesizing *intelligent hybrid systems* for modeling and control applications. The proposed framework integrates neural networks, fuzzy logic, and genetic algorithms in a single comprehensive paradigm, which consists of a generic architecture and a hybrid learning scheme. The developed paradigm maintains the following salient features:

- The system's architecture has a transparent structure. Its parameters, links, signals and modules have their own physical interpretations (meanings). This feature allows a better understanding and deeper insight into the system's behavior and provides easier debugging and troubleshooting during the design.

- The learning scheme is composed of three phases. The first phase performs a coarse identification for the system's numerical parameters using unsupervised learning (clustering) algorithms. The second phase finds the linguistic-association parameters (fuzzy rules) using unsupervised as well as supervised learning algorithms. In the third phase, the numerical parameters are optimized and fine-tuned using supervised learning and search techniques. This task decomposition makes the identification process systematic and more convenient.

- The proposed framework allows the incorporation of numerical input/output data and expert knowledge during the synthesis process. The learning algorithm in all its phases uses the available numerical data. Expert knowledge can help in finding the appropriate fuzzy partitioning for the input/output variables. Moreover, in the second phase, this knowledge plays an important role in forming and/or checking

the linguistic-fuzzy rules that are found by numerical algorithms. In the third phase, expert knowledge guides the formation of the fitness function of the MRD-GA, specially in control applications. In addition, it is utilized to choose the appropriate control-parameter values for different learning algorithms.

The above features have provided the proposed paradigm with many advantages; these advantages are:

- The system combines the merits of each individual technique. For example, it combines the learning ability in neural networks, the approximate and linguistic reasoning in fuzzy logic, as well as the optimization and search capabilities of genetic algorithms. Moreover, it overcomes the limitations of individual techniques.

- The system has the ability of incorporating qualitative as well as quantitative knowledge during learning. Furthermore, it is able to describe its decisions in an explicit natural language.

- The system preserves the model-free control feature that the classical fuzzy control offers. It does not need any information about the internal parameters of the plant under control.

- The system requires compact-sized data sets for the learning process, compared with what neural networks usually require [97]; this significantly decreases the learning-computation time.

- The system provides more robustness than, for example, classical fuzzy or neural-network systems, as the proposed system exploits all available sources of information in the synthesis procedure. In other words, the system utilizes available observations or experience which neural networks can't do. Furthermore, it utilizes the available input/output numerical data which classical fuzzy systems can't do either.

Many new algorithms have been proposed throughout the thesis. In addition, a number of well-known algorithms have been adapted to suit the assigned function in the synthesis process. The performance of these algorithms is outlined hereafter:

- In the first learning phase, the Kohonen's self-organizing feature maps and the fuzzy c-means clustering algorithm are adapted to perform coarse-identification for the numerical parameters and to find the most appropriate fuzzy partitioning.

Three clustering-validity measures have been used to assess the quality of clustering produced by the two algorithms. Several tests and evaluations [168] have shown that the fuzzy c-means clustering algorithm outperforms Kohonen's self-organizing feature maps.

- In the second learning phase, four different algorithms are presented. The competitive learning algorithm is adapted to find the linguistic-association parameters (fuzzy rules). Three algorithms are proposed as well; the minimum distance algorithm, the maximum matching-factor algorithm, and the static genetic algorithm. Evaluation studies [94] have shown that the static genetic algorithm provides the best performance among the four algorithms. However, because it is a supervised learning algorithm it tends to be much slower. The maximum matching factor algorithm has consistently performed better and faster than both competitive learning and minimum distance algorithms. Finally, the static genetic algorithm is recommended if accuracy is the only concern, and the maximum matching factor algorithm is recommended if a high performance per unit computational cost is required.

- In the third learning phase, the well-known backpropagation algorithm is adapted to optimize the numerical parameters initially identified in the first learning phase. Moreover, a multi-resolutional dynamic genetic algorithm is proposed for the same optimization task [93, 97]. A sensitivity analysis for the effect of different control-parameters on the performance of the multi-resolutional dynamic genetic algorithm has been carried out. Three parameters have been considered for this analysis; the population size, the crossover probability, and the type of the crossover operator. Population sizes in the range of 20~50 and crossover probabilities in the range of 0.5~0.9 are considered appropriate for most applications. The analysis indicates that the single-point crossover operator gives better performance than the double-point crossover operator. Compared to backpropagation algorithm [169], the multi-resolutional dynamic genetic algorithm tends to be more accurate but slower to converge. It also shows more cleverness at escaping from local minima and more flexibility in its application to modeling and control problems.

The effectiveness of the proposed *intelligent hybrid scheme* in modeling of complex systems, is assessed using two benchmarks [97]. The benchmarks are highly nonlinear and difficult to model using conventional methods. Detailed comparative studies with other modeling approaches that use intelligent techniques, have been carried out [97]. Compared with such approaches, the proposed scheme shows superior performance and obvious advantages. The accuracy of the proposed scheme

exceeds most of the other techniques while preserving the advantage of "linguistic modeling" that some other approaches do not have.

To investigate the effectiveness of the proposed *intelligent hybrid scheme* in nonlinear control, this is used in synchronous machine voltage-regulation and speed-stabilization studies. A neuro-fuzzy automatic voltage regulator is designed and tuned using the proposed scheme [87]. Extensive simulation studies show an improvement in the synchronous generator response from a learning phase to the next. Also, the simulation shows that the generator response, after learning has been completed, is much better in terms of settling times, over/under-shoots, and rising/falling times.

A neuro-fuzzy power system stabilizer for a single-machine infinite-bus system is synthesized using the proposed scheme [170]. Only 400 data samples are used in the synthesis process. The effectiveness of the neuro-fuzzy stabilizer is compared with that of a conventional power system stabilizer. Test results for various conditions show that the proposed neuro-fuzzy stabilizer has better performance than the conventional stabilizer. It provides good damping over a wide operating range and significantly improves the transient and dynamic performance of the synchronous machine, under different disturbances. The neuro-fuzzy stabilizer is less sensitive to changes in the system parameters than a conventional one.

Moreover, a neuro-fuzzy power system stabilizer for a multi-machine power system environment is designed and optimized using the proposed scheme [171]. A compact-sized training data of only 600 samples, is used in the design. Simulation results for various tests show that the proposed neuro-fuzzy stabilizer has better performance than that of a conventional stabilizer. It provides good damping for multi-mode oscillations under different disturbances, and significantly improves the transient and dynamic performance of synchronous machines in a multi-machine environment. Furthermore, the results show the good coordination and cooperation between conventional power system stabilizers and the proposed neuro-fuzzy stabilizer while working together in the same power system. Accordingly, power-system engineers are encouraged to upgrade the power systems with neuro-fuzzy stabilizers without the doubt of encountering performance or stability problems. .

9.2 Future work

Based on this thesis, some future research directions are suggested:

- Extending the sensitivity study to include other control parameters of the multi-resolutional dynamic genetic algorithm proposed in this thesis. The effect of the probability of mutation, the offsets (δ_m and δ_σ), and the decaying time constant (τ) can be examined in this study.

- Evaluating the employment of other heuristic search algorithms in the optimization phase of the learning scheme. Evolutionary programming techniques and evolutionary strategies are promising candidates for this study.

- Investigating the applicability of *intelligent hybrid systems* for on-line adaptation and real-time learning/control applications. There are two directions in this study. In the first direction, the adaptation process is performed for the linguistic-fuzzy rules and in this case the developed controller is called "self-organizing controller". In the other direction, the parameters of the membership functions (centers and widths) are adapted and in this case the developed controller is called "adaptive controller".

- Studying the hardware implementation issues for the proposed intelligent control scheme. This helps to give more insight into the applicability of such intelligent algorithms for industrial purposes.

- Investigating the applicability of the proposed learning scheme in other applications such as classification, pattern recognition, image processing, vision and robotics.

- Applying the proposed learning scheme in other power-systems applications such as load forecasting and fault diagnosis.

Appendix A

Synchronous-Machine Infinite-Bus Data

- **AVR Model:** $$E_f = \frac{K_{AVR}}{1 + sT_{AVR}}(V_{ref} - V_t + U_{pss})$$

- **Governor Model:** $$g = (a_g + \frac{b_g}{1 + sT_g})\dot{\delta}$$

- **Simulation Parameters:**

$X_d = 1.24$	$X_q = 0.7$	$R_{stator} = 0.007$
$X_{leakage} = 0.14$	$X_{field} = 1.33$	$R_{field} = 0.00089$
$K_{damping} = 0.027$	$H = 3.46$	$K_{AVR} = 200$
$T_{AVR} = 0.01$	E_f (upper limit) = 7 pu	E_f (lower limit) = -7 pu
$a_g = -0.001238$	$b_g = -0.17$	$T_g = 0.25$
$Z_{line-1} = 0.009+j\ 0.22$	$Z_{line-2} = 0.009+j\ 0.22$	$Z_{trasformer} = 0.01+j\ 0.12$

All resistances and reactances are in per-unit values, and time-constants are in seconds.

Appendix B

List of publications resulting from this thesis

[1] Wael A. Farag, V.H. Quintana, and G.L.-Torres, "A Genetic-Based Neuro-Fuzzy Approach for Modeling and Control of Dynamical Systems", accepted for publication in *IEEE Trans. on Neural Networks* (Special Issue on Hybrid Systems), expected May 1998.

[2] Wael A. Farag, V.H. Quintana, and G. Lambert-Torres, "An Optimized Fuzzy Controller for a Synchronous Generator in a Multi-Machine Environment", accepted for publication in *Fuzzy Sets and Systems* (Special Issue on Power Systems Applications), Elsevier Science, The Netherlands.

[3] Wael A. Farag, V.H. Quintana, and G.L. Torres, "Applications of Artificial Intelligence Techniques in Synchronous Machines Control: A Review", accepted for publication in the International Journal of *Engineering Intelligent Systems* (EIS), CRL Publishing Ltd., UK.

[4] Wael A. Farag, V.H. Quintana, and G.L.-Torres, "Neuro-Fuzzy Modeling of Complex Systems Using Genetic Algorithms", Proc. of the IEEE Inter. Conf. on Neural Networks (*IEEE ICNN'97*), Vol. 1, pp. 444-449, June 8-12, 1997, Houston, Texas, USA.

[5] W.A. Farag, V.H. Quintana, and G. Lambert-Torres, "Neural-Network-Based Self-Organizing Fuzzy-Logic Automatic Voltage Regulator for a Synchronous Generator", Proceeding of the 28th Annual *NAPS*, pp. 289-296, November 10-12, 1996, MIT, Cambridge, MA, USA.

[6] Wael Farag, V.H. Quintana, and G.L. Torres, "Genetic Algorithms and Back-propagation: A Comparative Study", accepted for publication in the proceedings of *IEEE Canadian Conference of Electrical and Computer Engineering*, Waterloo, Ontario, Canada, 24-28 May 1998.

[7] Wael A. Farag, V.H. Quintana, and G.L.-Torres, "Extraction of Fuzzy-Control Rules from the Input-Output Properties of a Controlled Plant", submitted journal paper.

[8] Wael A. Farag, V.H. Quintana, and G. Lambert-Torres, "On the Clustering Algorithms for Building Fuzzy Logic Systems", submitted journal paper.

[9] Wael A. Farag, V.H. Quintana, and G. Lambert-Torres, "A Genetic-Based Neuro-Fuzzy Controller for Synchronous Machines", submitted journal paper.

[10] Wael A. Farag, V.H. Quintana, and G. Lambert-Torres, "Enhancing Transient Stability of Multi-Machine Power Systems Using Intelligent Techniques", submitted conference paper.

Bibliography

[1] IEEE Transactions on Neural Networks, special issue on everyday applications of neural networks, Vol. 8, no. 4, July 1997.

[2] Gupta, M.M, "Survey of process control application of fuzzy set theory", Proceeding of the 18th IEEE Conference on Decision and Control, San Diego, January 1979.

[3] Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller", Int. J. Man-Machine Studies, 7(1), pp.1-13, 1975.

[4] L.P. Holmblad and J.J. Ostergaard, "Control of a cement kiln by fuzzy logic", In: Fuzzy Information and Decision Processes, Gupta M.M. and Sanchez E. (eds), North-Holland, Amsterdam, 1982.

[5] M. Sugeno and M. Nishida, "Fuzzy control of a model car", Fuzzy Sets and Systems 16, Elsevier Science Publishers B.V., pp. 103-113, 1985.

[6] K. Tanaka, M. Sano, and H. Watanabe, "Modeling and Control of Carbon Monoxide Concentration Using a Neuro-Fuzzy Technique", IEEE Transactions of Fuzzy Systems, Vol. 3, No. 3, August 1995.

[7] S. Yasunobu, S. Miyamoto, "Automatic train operation system by predictive fuzzy control", in M. Sugeno, ed., Industrial Applications of Fuzzy Control, Amesterdam, North-Holand, pp. 1-18, 1985.

[8] Momoh, X.W. Ma, K. Tomsovic, "Overview And Literature Survey Of Fuzzy Set Theory In Power Systems", IEEE Transactions on Power Systems, Vol. 10, No. 3, August 1995.

[9] D.A. Linkenes and H.D. Nyongesa, "Learning Systems in Intelligent Control: an appraisal of fuzzy, neural and genetic algorithm control applications", IEE Proc.-Control Theory Appl., Vol. 143, No. 4, July 1996.

[10] L. A. Zadeh, "Fuzzy sets," Information and Control, Vol. 8, pp. 338-353, 1965.

[11] Zadeh L., "Outline of a new approach to analysis of complex systems and decision processes", IEEE Trans. on Systems, Man, and Cybernetics, SMC-3, pp. 28-44, 1973.

[12] Chuan-Chang Hung, "Building a Neuro-Fuzzy Learning Control Systems", AI Expert, November 1993.

[13] K.J. Astrom and T.J. McAvoy, "Intelligent control: an overview and evaluation", in White and Sofge, 'Handbook of intelligent control: neural, fuzzy adaptive approaches' (Van Nostrand Reinhold, NY, 1992), pp. 3-34.

[14] "Seeking the essence of intelligent control", IEEE Control Systems Mag., June 1994.

[15] "Intelligence and learning", IEEE Control Systems Mag., June 1995.

[16] J.J. Buckely and Y. Hayashi, "Fuzzy neural networks - a survey', Fuzzy Sets and Systems, 1994, Vol. 66, no. 1, pp. 1-13.

[17] C.C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I", IEEE Trans. on Systems Man and Cybernetics, Vol. 20, No. 2, March 90, pp. 404-418.

[18] C.C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part II", IEEE Trans. on Systems Man and Cybernetics, Vol. 20, No. 2, March 90, pp. 419-435.

[19] T. Takagi and M. Sugeno, "Fuzzy Identification of Systems and Its Applications to Modeling and Control", IEEE Transactions on systems, man, and cybernetics, vol. 15, no. 1, 1985.

[20] Richard P. Lippmann, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, April 1987, pp. 4-22.

[21] Panos J. Antsaklis, "Neural Networks in Control Systems", IEEE Control Systems Magazine, April 1990.

[22] K.S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks" IEEE Transactions on Neural Networks, Vol. 1, No. 1, March 1990.

[23] Chris Barnes, "Applications of Neural Networks to Process Control and Modeling", IEEE Conference on Neural Networks, 1991.

[24] Psaltis, A. Sideris, and A. Yamamura, "A Multilayered Neural Network Controller", IEEE Control Systems Magazine, April 1988.

[25] Y. El-Haddad, "Adaptive Control using Neural Networks", M.Sc. thesis, Ain-Shams University, Computer and Systems Engineering Department, Cairo,1995.

[26] Saerens and A. Soquet, "Neural Network Controller Based on Back-Propagation Algorithm", IEE Procedings-F, Vol. 138, No. 1, May 1991.

[27] K.J. Hunt, D. Sbarbaro, R. Zbikowski, and P.J. Gawthrop, "Neural Networks for control systems - a survey", Automatica, Vol. 28, 1992, no. 6, pp. 1083-1112.

[28] P.J. Werbos, "Neuro-control and supervised learning: an overview and evaluation' in White and Sofge, 'Handbook of intelligent control: neural, fuzzy adaptive approaches' (Van Nostrand Reinhold, NY, 1992), pp. 3-34.

[29] D.H. Nguyen, and B. Widrow, "Neural Networks for self-learning control systems", International Journal of Control, 1991, Vol. 54, no. 6, pp. 1439-1452.

[30] A. G. Batro, R.S. Sutton, and C.H. Anderson, "Neurolike adaptive elements that can solve difficult learning control problems", IEEE Trans. on Systems, Man, and Cybernetics, Vol. 13, no. 5, pp. 834-846.

[31] D. E. Goldberg, "Genetic Algorithms is Search, Optimization, and Machine Learning", Reading, MA: Addison-Wesley, 1989.

[32] Brindle A.,"Genetic Algorithms for Function Optimization", Ph.D. thesis, CS dept., The University of Alberta, Edmonton, Alberta, Canada.

[33] Y. Davidor, "Genetic Algorithms and Robotics, A Heuristic Strategy for Optimization", 1991 World Scientific Publishing Co. Pte. Ltd., ISBN 9810202172.

[34] Z. Miachalewicz, "GA + Data Structure = Evolution Programming", New York: Springer-Verlag, 1994, ISBN 3-540-58090-5.

[35] J. H. Holland, "Adaptation in Natural and Artificial Systems", Ann Arbor, MI: University of Michigan, 1975.

[36] J.J. Choi, P. Arabshahi, R.J. Marks, and T.P. Caudell, "Fuzzy parameter adaptation in neural systems", Proceedings of international joint conference on Neural Networks, 1992, pp. 232-235.

[37] L.X. Wang, and J.M. Mendel, "Fuzzy basis functions, universal approximations and orthogonal least squares", IEEE Trans. on Neural Nets., 1992, pp. 807-814.

[38] S. Horikawa, T. Furuhashi, S. Ouma, and Y. Uchikawa, "A fuzzy controller using a neural network and its capability to learn expert's control rules", Proceedings of international conference on Fuzzy logic and Neural Networks, 1990, Vol. 1, pp. 103-106.

[39] H. Takagi, N. Suzuki, T. Koda, and Y. Kojima, "Neural networks designed on approximate reasoning architecture and thier applications", IEEE Trans. Neural Networks, Vol. 3, no. 5, 1992, pp. 752-760.

[40] Chen-Teng Lin, C. S. George Lee, "Neural-Network-Based Fuzzy Logic Control and Decision System", IEEE Transactions on Computers, Vol. 40, No. 12, December 1991.

[41] H. Takagi and I. Hayashi, "NN-driven fuzzy reasoning", Inter. Journal Approx. Reasoning, 1991, Vol. 5, no. 3, pp. 191-212.

[42] H.R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcement", IEEE Trans. Neural Networks, 1992, Vol. 3, no. 5, pp. 724-740.

[43] S.G. Kong and B. Kosko, "Adaptive fuzzy systems for backing up a truck-and-trailer", IEEE Trans. Neural Nets, 1992, Vol. 3, no. 2, pp. 211-223.

[44] F. Bouslama, and A. Ichikawa, "Application of neural networks to fuzzy control", Neural Networks, 1993, Vol. 6, pp. 791-799.

[45] S. Horikawa, T. Furuhashi, Y. Uchikawa, and T. Tagawa, "A study of fuzzy modeling using fuzzy neural networks", Proceedings of international symposium on Fuzzy engineering, 1991.

[46] C.M. Higgins and R.M. Goodman, "Fuzzy rule-based networks for control", IEEE Trans. Fuzzy Systems, 1994, Vol. 2, no. 1, pp. 82-88.

[47] T. Kohonen, "Self-organization and associative memory", Springer-Verlag, Berlin, 1984.

[48] C.T. Lin and C.S.G. Lee, "Reinforcement structure/parameter learning for neural network based fuzzy logic control systems", Proceedings of IEEE international conference on fuzzy systems, San Francisco, CA, USA, 1993, pp. 88-93.

[49] J. Nie and D.A. Linkens, "Learning control using fuzzified self-organizing radial basis function network", IEEE Trans. Fuzzy Systems, 1993, Vol. 1, no. 4, pp. 280-287.

[50] J. Nie and D.A. Linkens, "FCMAC: A fuzzified cerebellar model articulation controller with self-organizing capacity", Automatica, 1994, Vol. 30, no. 4, pp. 655-664.

[51] K.H. Kyung and B.H. Lee, "Fuzzy data-base derivation using neural-network based fuzzy logic controller by self-learning", Proceedings of IECON, 1993, pp. 435-440 (Part I).

[52] J.-S. R. Jang, "ANFIS: Adaptive-Network-Based Fuzzy Inference System", IEEE Trans. on Systems, Man, and Cybernetics, Vol. 23, No. 3, pp. 665-685, 1993.

[53] C. L. Karr and E.J. Gentry, "Fuzzy Control of pH Using Genetic Algorithms", IEEE Transactions Fuzzy Systems, vol. 1, no. 1, pp. 46-53, 1993.

[54] C.L. Karr and J. Schaffer, "Design of an adaptive fuzzy logic using a genetic algorithms", Proceedings of international conference on genetic algorithms, 1989, (Morgan Kaufmann).

[55] C.L. Karr, R. Belew, and L. Booker, "Genetic Algorithm based fuzzy control of spacecraft autonomous rendezvous", Proceedings of international conference on Genetic Algorithms, (1991), (Morgan Kaufmann).

[56] D.A. Linkens and H.O. Nyongesa, "Genetic Algorithms for fuzzy control part 1: On-line system development and application", IEE Proc. Control Theory Appl., Vol. 142, no. 3, pp. 161-176, 1995.

[57] D.A. Linkens and H.O. Nyongesa, "Genetic Algorithms for fuzzy control part 2: On-line system development and application", IEE Proc. Control Theory Appl., Vol. 142, no. 3, pp. 177-185, 1995.

[58] M.A. Lee and H. Takagi, "Integrating design stage of fuzzy systems using genetic algorithms", Proc. of IEEE international conference on Fuzzy Systems, San Francisco, CA, USA, 1993, pp. 612-617.

[59] Abdollah Homaifar and Ed McCormick, "Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers Using Genetic Algorithms", IEEE Transactions on Fuzzy Systems, Vol. 3, No. 2, May 1995.

[60] D. Park, A. Kandel, and G. Langholz, "Genetic-Based New Fuzzy Reasoning Models with Application to Fuzzy Control", IEEE Transactions on Systems, Man, and Cypernetics, Vol. 24, No. 1, January 1994.

[61] T. Shibata and T. Fukuda, "Coordinative behavior by genetic algorithm and fuzzy in evolutionary multi-agent system", Proceedings of IEEE international conference on Robotics and automation, Atlanta, GA, USA, May 1993, pp. 760-765.

[62] H. Ishibuchi, N. Yamamoto, T. Murata and H. Tanaka, "Genetic Algorithms and nieghbourhood search algorithms for fuzzy flowshop scheduling problems", Fuzzy sets and Systems, 1994, Vol. 67, no. 1, pp. 81-100.

[63] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Selecting Fuzzy If-Then Rules for Classification Problems Using Genetic Algorithms", IEEE Transactions on Fuzzy Systems, Vol. 3, No. 3, August 1995.

[64] J.D. Schaffer, D. Whitley, and L.J. Eshelman, "Combinations of genetic algorithms and neural networks: a survey of the state-of-the-art", Proceedings of international workshop on Combination of GA and NN, Baltimore, MD, USA, 1992, pp. 1-37.

[65] V. Petridis, S. Kazarlis, A. Papaikonomou, and A. Filelis, "A hybrid genetic algorithm for training neural networks", Proceedings of international conference on Artificial neural networks, Brighton, UK, 1992, pp. 953-956.

[66] S.L. Lyung and H. Adli, "A parallel genetic/neural network learning algorithm for MIMD shared-memory machines", IEEE Trans. Neural Networks, 1994, Vol. 5, no. 6, pp. 900-909.

[67] R.J. Machdo, A.F. Da Rocha, "Evolutive neural networks", IEEE inter. conf. on Fuzzy Systems, San Diego, CA, USA, 1992, pp. 493-500.

[68] J.R. Koza and J.P. Rice, "Genetic generation of both the weights and architecture of a neural network", Proceedings of inter. joint conf. on neural networks, Seattle, WA, USA, 1991, Vol. 2, pp. 397-404.

[69] D. Dasgupta and D.R. McGregor, "Designing neural networks using the structured genetic algorithm", Proceedings of inter. conf. on artificial neural networks, Brighton, UK, 1992, pp. 263-268.

[70] D.A. Linkens and H.O. Nyongesa, "Evolutionary learning in fuzzy neural control", Proc. Of 3rd European congress on Intelligent techniques and soft computing, EUFIT '95, Aachen, Germany, 28-31, August 1995.

[71] D.S. Feldman, "Fuzzy network synthesis with genetic algorithms", Proc. of Inter. Conf. On Fuzzy Systems, San Diego, CA, USA, 1992, pp. 493-500.

[72] D. Cliff, I. Harvey, P. Husbands, "Incremental evolution of neural network architectures for adaptive behavior", Proc. of European Symp. on Artif. Neural networks, 1993, pp. 39-44.

[73] R.M. Tong, "The construction and evaluation of fuzzy models", In: Advances in Fuzzy Set Theory and Applications, M.M. Gupta, R.K. Ragade and R.R. Yager, (eds.), North Holand, 1979, pp. 559-576.

[74] W. Pedrycz, "An identification algorithm in fuzzy relational systems", Fuzzy Sets Sys., Vol. 13, pp. 153-167, 1984.

[122] E.V. Larsen and D.A. Swann, "Applying power system stabilizers, Part III: Practical considerations", IEEE Trans. on PAS, vol. 100, No. 6, pp. 3047-3054, June 1981.

[123] F.R. Schlief, H.D. Hunkins, G.E. Martin, and E.E. Hattan, "Excitation control to improve power system stability", IEEE Trans. on PAS, vol. 87, pp. 1426-1434, June 1968.

[124] P.L. Dandeno, A.N. Karas, K.R.McClymont, and W. Watson, "Effect of high speed rectifier excitation system on generator stability limits", IEEE Trans. on Power Apparatus and Systems, vol. PAS-87, pp. 190-200, January 1968.

[125] Yao-Nan Yuand, and H.A.M. Moussa, "Optimal stabilization of a multi-machine system", IEEE Trans. on PAS, vol. PAS-91, No. 3, pp. 1174-1182, May/June 1972.

[126] K.Ohtsuka, S. Yokokama, H. Tanaka, and H. Doi, "A multivariable optimal control system for a generator", IEEE Trans. On Energy Conversion vol. EC-1, pp. 88-89, June 1986.

[127] M.M. Elmetwally, N.D. Rao, and O.P. Malik, "Experimental results on the implementation of an optimal control for synchronous machines", IEEE Trans. on Power Apparatus and Systems, vol. PAS-94, pp. 1192-1200, July/August 1975.

[128] A. Ghosh, G. Ledwich, O.P. Malik, and G.S. Hope, "Power system stabilizer based on adaptive control techniques", IEEE Trans. on PAS, vol. 103, pp. 1983-1989, August 1984.

[129] S.J. Cheng, O.P. Malik, and G.S. Hope, "Damping of multimode oscillations in power system using a dual-rate adaptive stabilizer", IEEE Trans. on PAS vol. 3, pp. 101-108, Feb. 1988.

[130] G.P. Chen, O.P. Malik, G.S. Hope, Y.H. Qin, and G.Y. Xu, "An adaptive power system stabilizer based on the self-optimizing pole shifting control strategy", in IEEE/PES 1993 winter meeting, Columbus, Ohio, January 31- February 5, 1993.

[131] O.P. Malik, G.P. Chen, G.S. Hope, Y.H. Qin, and G.Y. Xu, "Adaptive self-optimizing pole shifting control algorithm", IEEE Trans. on Energy Conversion, Vol. 8(4), pp. 639-645, 1993.

[132] C. Sun, Z. Zhao, Y. Sun, and Q. Lu, "Design of nonlinear robust excitation control for multimachine power systems", IEE Proceedings, Generation, Transmission Distribution, Vol. 143, No. 3, May 1996, pp. 253-257.

[133] S. Chen and O.P. Malik, "Power system stabilizer design using mu synthesis", IEEE Transactions on Energy conversion, 1994.

[134] S. Chen, O.P. Malik, "H_∞ optimization-based power system stabilizer design", IEE Proc.-Gener., Transm., Distrib., Vol. 142, No. 2, March 1995, pp. 179-184.

[135] Y. Akimoto, H. Tanaka, J. Yoshizawa, D.B. Klapper, W.W. Price, and K. A. Wirgau, "Transient Stability Expert System", IEEE Trans. on Power Systems, Vol. 4, No. 1, Feb. 1989, pp. 312-320.

[136] L. Wehenkel, Th. Van Cutsem, and M. Ribbenes-Pavella, "An Artificial Intelligence Framework for on-line Transient Stability Assessment of Power Systems", IEEE Transactions on Power Systems, Vol. 4, No. 2, May 1989, pp. 789-797.

[137] T. Hiyama, "Application of rule-based Stabilizing Controller to Electrical Power System", IEE Proceedings, Vol. 136, Pt. C, No. 3, May 1989, pp. 175-181.

[138] Y.Y. Hsu and C.R. Chen, "Tuning of Power System Stabilizers Using An Artificial Neural Networks", IEEE Transactions on Energy Conversion, Vol. 6, No. 4, pp. 612-619, Dec. 1991.

[139] Q.H. Wu, B. W. Hogg, and G. W. Irwin, "A Neural Network Regulator for Turbogenerators", IEEE Transactions on Neural Networks, Vol. 3, No. 1, January 1992.

[140] Diane C. Kennedy, Victor H. Quintana, "Neural Networks Regulators for Synchronous Machines", Proc. of ESAP 93, pp. 531-535, Melbrourne, Australia, January 1993.

[141] Diane C. Kennedy, Victor H. Quintana, "Power System Dynamic Control Using Neural Networks", IFAC 12th World Congress, Vol. 5, pp. 131-136, Sydney, Australia, July 1993.

[142] Zhang, O.P. Malik, G.S. Hope, G.P. Chen, "Application of an Inverse Input/Output Mapped ANN as a Power System Stabilizer", IEEE Trans. on Energy Conversion, Vol. 9, No. 3, pp. 433-440, September 1994.

[143] Y. Zhang, O.P. Malik, and G.P. Chen, "Artificial Neural Networks Power Systems Stabilizers in Multi-Machine Power System Environment", IEEE Transactions on Energy Conversion, Vol. 10, No. 1, pp. 147-155, March 1995.

[144] L. Guan, S. Cheng, and R. Zhou, "Artificial neural network power system stabilizer trained with an improved BP algorithm", IEE Proc.-Gener. Transm. Distrib., Vol. 143, No. 2, Mar 96.

[145] M.A. Abido and Y.L. Abdel-Magid, "Radial basis function network based power system stabilizers for multimachine power systems", Proc. Inter. Conf. On Neur. Net., IEEE ICNN'97, Vol. 2, pp. 622-626, Houston, Texas, USA, June 8-12, 1997.

[146] Y.M. Park, and K.Y. Lee, "A Neural Network-Based Power System Stabilizer using Power Flow Characteristics", IEEE Trans. on Energy Conv., Vol. 11, No. 2, pp. 435-41, June 96.

[147] Y.Y. Hsu, and C.H. Cheng, "Design of Fuzzy Power System Stabilizers for Multi-machine Power Systems", IEE Proceedings, Vol. 137, part C, No. 3, May 1990, pp. 233-238.

[148] M.A.M Hassan, O.P. Malik and G.S. Hope, "A Fuzzy Logic Based Stabilizer for A Synchronous Machine", IEEE Trans. on EC, Vol. 6, No. 3, pp. 407-413, Sep. 1991.

[149] M.A.M. Hassan, and O.P. Malik, "Implementation and Laboratory Test Results For Fuzzy-Logic Based Self-Tuned Power System Stabilizer", IEEE Transactions on Energy Conversion, Vol. 8, No. 2, pp. 221-228, June 1993.

[150] Takashi Hiyama, "Real Time Control of Micro-machine System using Micro-computer based Fuzzy Logic Power System Stabilizer", IEEE Transactions on Energy Conversion, Vol. 9, No. 4, pp. 724-731, December 1994.

[151] K.A. El-Metwally, and O.P. Malik, "Fuzzy Logic Power System Stabilizer", IEE Proceeding of Gener. Transm. Distrib., Vol. 142, No. 3, May 1995, pp. 277-281.

[152] K.A. El-Metwally, and O.P. Malik, "Application of Fuzzy Logic Stabilizers in a Multi-machine Power System Environment", IEE Proc.-Gener. Transm. Distrib., Vol. 143, No. 3, May 1996, pp. 263-268.

[153] K.A.M. El-Metwally, G.C. Hancock and O.P. Malik, "Implementation of a Fuzzy Logic PSS Using a Micro-controller and Experimental Test Results", IEEE Transactions on Energy Conversion, Vol. 11, No. 1, pp. 91-96, March 1996.

[154] P. Hoang, and K. Tomsovic, "Design and Analysis of an Adaptive Fuzzy Power System Stabilizer", IEEE Trans. on Energy Conversions, Vol. 11, No. 2, pp. 455-461, June 1996.

[155] A.R. Hasan, T.S. Martis and A.H.M.S. Ula, "Design and Implementation of a Fuzzy Controller Based Automatic Voltage Regulator for a Synchronous Generator", IEEE Transactions on Energy Conversion, Vol. 9, No. 3, pp. 550-557, September 1994.

[156] Takashi Hiyama, "Robustness of Fuzzy Logic Power System Stabilizers Applied to Multimachine Power System", IEEE Transactions on Energy Conversion, Vol. 9, No. 3, pp. 451-459, September 1994.

[157] P.K. Dash, and A.C. Liew, "Anticipatory fuzzy control of power systems", IEE Proceedings of Gener. Transm. Distrib., Vol. 142, No. 2, March 1995, pp. 211-218.

[158] Y.M. Park, and U.C. Moon and K. Y. Lee, "A Self-Organizing Power System Stabilizer using Fuzzy Auto-Regressive Moving Average (FARMA) Model", 037-2 Energy Conversion, IEEE/Winter Meeting 1996.

[159] H.A. Toliyat, Javad Sadeh and Reza Ghazi, "Design of Augmented Fuzzy Logic Power System Stabilizers to Enhance Power Systems Stability", IEEE Transactions on Energy Conversion, Vol. 11, No. 1, March 1996, pp. 97-103.

[160] T. Hiyama, K. Miyazaki, "A Fuzzy Logic Excitation System for Stability Enhancement of Power Systems with Multi-mode Oscillations", IEEE Transactions on Energy Conversion, Vol. 11, No. 2, pp. 449-454, June 1996.

[161] V.H. Quintana and S. Tyc, "Linear Optimal Regulator Based On Fuzzy-Logic Neural-Networks", International Conference on ISAP'94, Montpellier, France, 5-9 Sept. 94.

[162] H.C. Chang, and M.H. Wang, "Neural Network-Based Self-Organizing Fuzzy Controller for Transient Stability of Multimachine Power Systems", IEEE Transactions on Energy Conversion, Vol. 10, No. 2, pp. 339-346, June 1995.

[163] A. Hariri, and O.P. Malik, "A fuzzy logic based power system stabilizer with learning ability", 1996 IEEE/PES Winter Meeting, Jan 21-25, 1996, Baltimore, MD.

[164] J. Wen, S. Cheng, and O.P. Malik, "A Synchronous Generator Fuzzy Excitation Controller optimally Designed with A Genetic Algorithm", Proc. Of the 20th Inter.

Conf. On Power Ind. Comp. Applications (PICA), May 11-16, 1997, Columbus, Ohio, pp. 106-111.

[165] T. Senjyu, A. Miyazato, K. Uezato, and H.K. Hee, "Improvement of Multi-Machine Power System Stability by Cooperative Control of AVR and GOV Using Fuzzy-Genetic Controller", Proc. of the Inter. Conf. on Intell. Sys. Appl. to Power Sys. (ISAP'97), July 6-10, 1997, Seoul, Korea, pp. 89-93.

[166] Wael A. Farag, V.H. Quintana, and G.L. Torres, "Applications of Artificial Intelligence Techniques in Synchronous Machines Control: A Review", accepted for publication in the International Journal of Engineering Intelligent Systems (EIS), CRL Publishing Ltd., UK.

[167] M. Abdelnour, C.H. Chang, F.H. Huang, and J.Y. Cheung, "Design of a Fuzzy Controller Using Input and Output Mapping Factors", IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 5, Sept./Oct. 1991.

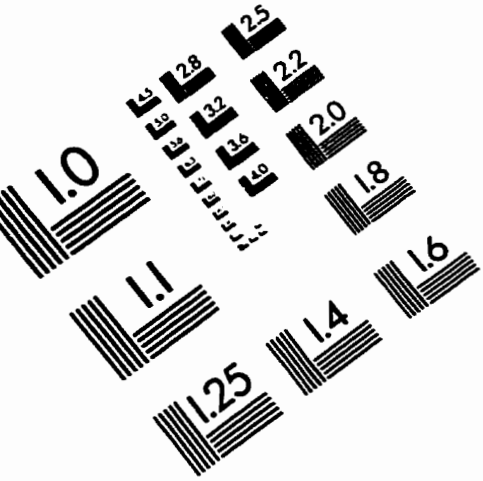
[168] Wael A. Farag, V.H. Quintana, and Lambert-Torres, "On the Clustering Algorithms for Building Fuzzy Logic Systems", submitted journal paper.

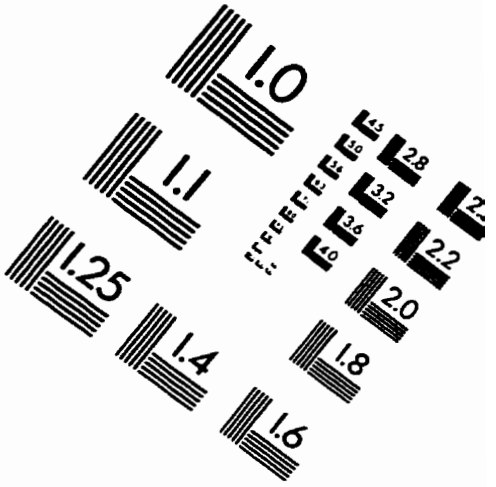
[169] Wael Farag, V.H. Quintana, and G.L. Torres, "Genetic Algorithms and Back-propagation: A Comparative Study", accepted for publication in the proceedings of *IEEE Canadian Conference of Electrical and Computer Engineering*, Waterloo, Ontario, Canada, 24-28 May 1998.

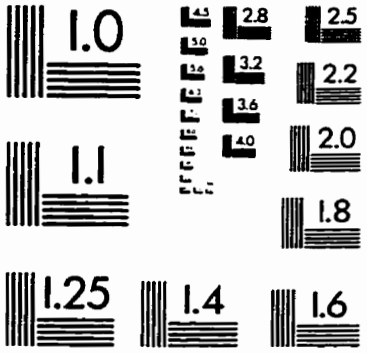
[170] Wael A. Farag, V.H. Quintana, and G. Lambert-Torres, "A Genetic-Based Neuro-Fuzzy Controller for Synchronous Machines", submitted journal paper.

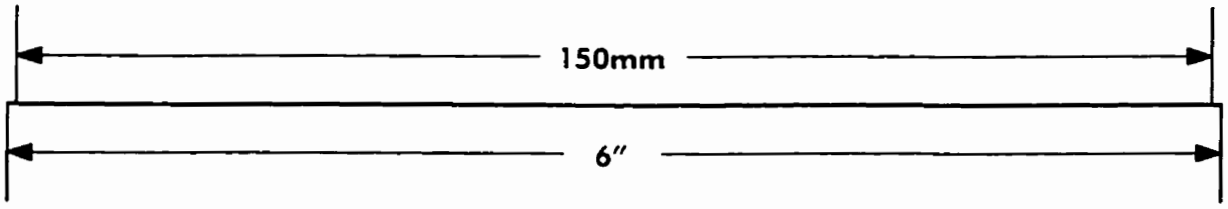
[171] Wael A. Farag, V.H. Quintana, and Lambert-Torres, "An Optimized Fuzzy Controller for a Synchronous Generator in a Multi-Machine Environment", accepted for publication in *Fuzzy Sets and Systems* (Special Issue on Power Sysems Applications), Elsevier Science, The Netherlands.

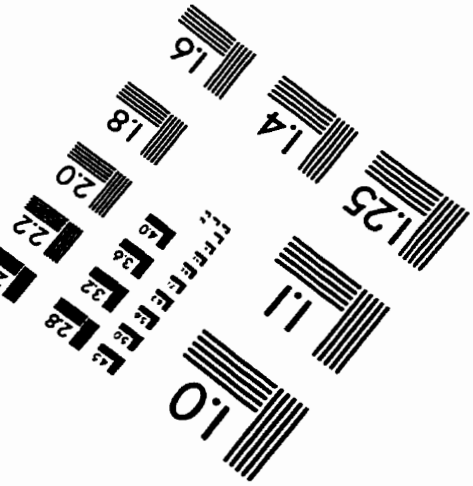
IMAGE EVALUATION
TEST TARGET (QA-3)

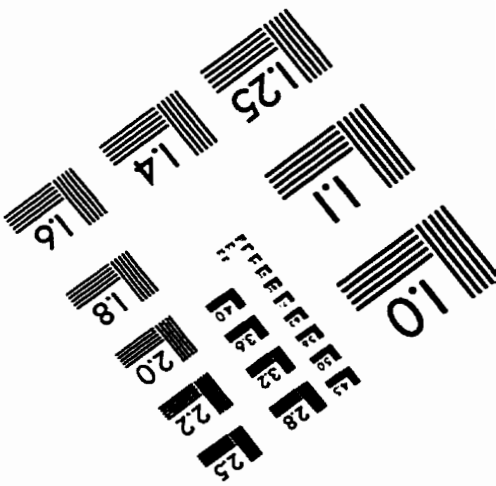












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989