Fuzzy Logic and Soft Computing:

Technology Development and Applications

by

Piero P. Bonissone

General Electric CRD

Schenectady NY 12309, USA.

March 12, 1997

Contents

1 Summary

1.1 Objective

The purpose of these notes is to introduce the reader to Fuzzy Logic based controllers or Fuzzy Controllers (FCs), one of the most promising emerging technology in the field of Engineering.

First, we will focus on the technology development of Fuzzy Controllers. We will explain their development, compilation, and deployment process, and we will discuss a representative sample of their industrial applications.

Then, we will discuss Fuzzy Controllers within the context of the broader field of Soft Computing (SC), a new discipline that combines emerging problem-solving technologies such as Fuzzy Logic (FL), Probabilistic Reasoning (PR), Neural Networks (NNs), and Genetic Algorithms (GAs). Each of these technologies provide us with complementary reasoning and searching methods to solve complex, real-world problems.

Within this broader context, we will analyze and illustrate some of the most useful combinations of SC components, such as the use of FL to control GAs and NNs parameters; the application of GAs to evolve NNs (topologies or weights) or to tune FL controllers; and the implementation of FL controllers as NNs tuned by backpropagation-type algorithms.

1.2 Motivation

The complexity of real-world engineering problems exemplifies the famous *principle of incompatibility*, presented by Zadeh in 1973 (Zadeh, 1973). In his observations Zadeh statd that: *" ... as the complexity of a system increases, our ability to make precise and yet significant statements about the its behavior diminishes, until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics."* From this principle and from other observations, we can conclude that:

- *There is a tradeoff between precision and reliability.* Our quest for precision is usually incompatible with our quest for reliability and robustness: as we strive to provide very precise answers to ill-defined and complex problems we introduce unrealistic assumptions in our models that cause such a precise answer not to be the true one.

- *There is a tradeoff between precision and complexity.* The complexity required to solve a given problem increases considerably with the required solution's precision. Common tasks routinely performed by humans, such as parking an automobile parallel to the curb, quickly become untractable, and therefore impossible to be performed in real time by a machine, as the problem requirements become too precise, e.g: backing-up at -2.75 mph, counterteering at not more than 53.5 degrees, leaving the car at a distance of 2.45 inches from the curb, etc.)

The key issue is to determine the *information granularity* required to exploit the problem's intrinsic tolerance for imprecision, thus providing us with an imprecise answer that is still useful, reliable, and inexpensive to compute. The ability to reason with fuzzy granules (via deductions and interpolation mechanisms) is one of the strongest points of Fuzzy Systems.

In this report we will focus on the use of Fuzzy Systems to synthesize controllers for dynamical systems.

1.3 Structure of the Report

This report is subdivided into three major parts: FLC technology, FLC applications, and FLC within the scope of Soft Computing. In the first part we will illustrate the development process common to Fuzzy Systems (both fuzzy rule based systems and fuzzy controllers). This will be followed by a detailed description of fuzzy controllers technology and its components: FLC interpreters, compilers, and run-time engine. A detailed description can be found in (Bonissone, 1991b, Bonissone and Chiang, 1993).

In the second part, we will discuss some of the FLC applications. We will compare these applications in a cost/complexity framework, and examine the driving factors that led to the use of FLCs in each application. We will emphasize the role of fuzzy logic in developing supervisory controllers and in maintaining explicit the tradeoff criteria used to manage multiple control strategies. See (Bonissone *et al.*, 1995, Bonissone, 1994).

Finally in the third part of this report we will show how FLC and in general Fuzzy Systems are components of a broader paradigm called Soft Computing (SC). After a brief description of each of SC technologies, we will analyze some of their most useful combinations, such as the use of FL to control GAs and NNs parameters; the application of GAs to evolve NNs (topologies or weights) or to tune FL controllers; and the implementation of FL controllers as NNs tuned by backpropagation-type algorithms. See(Bonissone, 1997).

2 PART I: Fuzzy Logic Controllers

2.1 FLC Introduction

Over the last decade, the number of applications of Fuzzy Logic Controllers (FLCs) has dramatically increased. Initially outlined by Zadeh (Zadeh, 1973) and explored by Mamdani (Mamdani and Assilian, 1975, Kickert and Mamdani, 1978) in the early seventies, FLC applications exhibited their first industrial and commercial growth in Japan almost a decade later (Sugeno, 1985, Yasunobu and Miyamoto, 1985). Since then, many Japanese companies have offered consumer-oriented products enhanced by FLC technology, such as Canon's camera autofocus control (Shingu and Nishimori, 1989), Honda's and Nissan automatic transmission (Takahashi *et al.*, 1991), Mitsubishi's room air conditioner control, Panasonic's clotheswasher control, and Toshiba's elevator control.

With a better understanding of FLC synthesis and analysis methodologies and the availability of commercial FLC development tools, FLC applications have also become more popular in the US and Europe. In this paper we will focus on the development and deployment of five FLC technology applications, describe the enabling FLC technology and offer some predictions of future research trends.

We will briefly address the problem of synthesizing nonlinear controllers and describe the typical fuzzy logic based solution. Then we will provide a comparison framework defined by development and deployment cost, required performance, and throughput.

2.2 Synthesizing the Control Surface

The fuzzy Proportional Integral (PI) controller is one of the most common fuzzy logic controllers. This controller is defined by a customized nonlinear control surface in the $(e, \dot{e}, du)$ space. In this section we will compare the fuzzy PI with some of its conventional counterparts, namely the conventional PI and the two dimensional sliding mode controller.

2.2.1 Conventional PI controllers

Conventional controllers are derived from control theory techniques based on mathematical models of the open-loop process to be controlled. For instance, a conventional proportional-integral (PI) controller can be described by the function

$$\begin{aligned} u &= K_p e + K_i \int e dt \\ &= \int (K_p \dot{e} + K_i e) dt \end{aligned}$$

or by its differential form

$$du = (K_p \dot{e} + K_i e) dt$$

The proportional term provides control action equal to some multiple of the error, while the integral term forces the steady state error to zero. Otherwise, the controller will always force a change in the manipulated variable.

Geometric Interpretation of a PI In the case of the conventional PI controller, for instance, let e be defined as the set point subtracted from the actual value of a given signal, and let positive $\dot{e}$ denote an increasing rate of change of e. Assume a control law that requires a high positive du to counteract a high negative e with a high negative $\dot{e}$ and a high negative du to counteract a high positive e with a high positive $\dot{e}$. Also assume that the goal of the control law is to bring the system to the equilibrium point of zero e and zero $\dot{e}$. In a three dimensional space with axes e, $\dot{e}$, and du, the control surface du of a conventional PI would be a **plane** passing through the origin and oriented at some angle with respect to the e-$\dot{e}$ plane, the angle determined by the particular values of K_p and K_i, as shown in Figure 1 (a).

Linear controllers, which are relatively easy to develop, can be visualized as hyperplanes in an $(n+1)$ dimensional space, mapping an n dimensional state vector to a control action. Nonlinear controllers, on the other hand, represent a much harder synthesis problem. The intrinsic difficulty of this task has spurred the development of alternative control synthesis techniques, such as Fuzzy Logic Control.

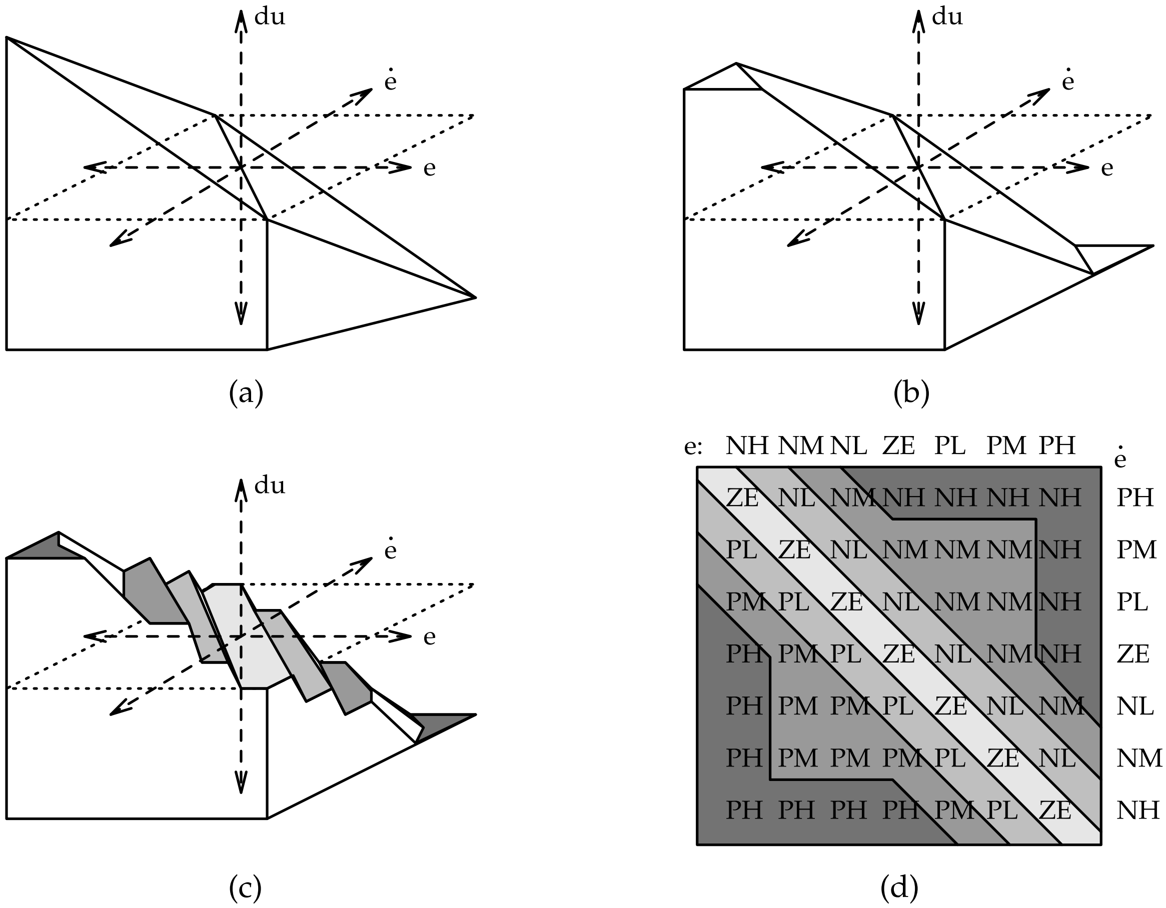

Figure 1: The control surface of: (a) a conventional PI controller; (b) a two dimensional sliding mode controller; (c) a fuzzy logic PI controller; and (d) the rule set defining the fuzzy PI's surface.

Tuning PIs Once initial values of K_p and K_i have been determined by the Zeigler-Nichols method, a number of heuristics are used to fine tune those values. Increasing K_p causes the rise time to decrease, because the error will be amplified and the controller will output a greater controller action. However, a large increase in K_p will also cause the controlled variable to overshoot its steady state value, and the oscillations about that value to markedly increase. Decreasing K_i will reduce the overshoot of the controlled variable at the expense of the rise time, because the integral of the error will be attenuated.

2.2.2 Fuzzy Logic PI Controllers

FLCs are knowledge based controllers usually derived from a knowledge acquisition process or automatically synthesized from self-organizing control architectures (Procyk and Mamdani, 1979). These controllers typically define a nonlinear mapping from the system's state space to the control space. Thus each FLC can be visualized as a nonlinear control surface reflecting a process operator's or a product engineer's prior knowledge. Each control surface is declaratively represented in a Knowledge Base (KB) and executed by an interpreter or a compiler.

The KB consists of a set of fuzzy rules, termsets, and scaling factors, which are evaluated by an interpreter. For instance, in the case of a fuzzy proportional-integral (PI) controller, the rule set maps linguistic descriptions of state vectors $[e, \dot{e}]$ into incremental control actions du; the termsets define the semantics of the linguistic values used in the rules; and the scaling factors determine the extremes of the numerical range of values for both the input and output variables. Using fuzzy logic we can synthesize a step-like control surface with gradations between the steps (obtained from an interpolation mechanism), which generalizes the control surface of the conventional PI, as can be seen in Figure 1 (c).

Reference fuzzy sets are defined for e, $\dot{e}$, and du in their corresponding termsets. Similarly, scaling factors N_e, $N_{\dot{e}}$, and N_{du} are also defined to determine the range of values for e, $\dot{e}$, and du, e. g. $-N_e \leq e \leq N_e$. In the rule set, a distribution for the controller output du is defined for each combination of the linguistic sets for e and $\dot{e}$, as illustrated in Figure 1 (d). In the figure, the error e has been divided into seven fuzzy sets; PH is positive high, PM positive medium, PL positive low, ZE zero, NL negative low, NM negative medium, and NH negative high. $\dot{e}$ has also been divided into the reference fuzzy sets with the same linguistic labels. It is important to note that e and $\dot{e}$ are not defined over the same universe of discourse, so their membership functions need not be identical. The rules have an intuitive interpretation. For example, if e has a negative medium value and $\dot{e}$ has a negative low value, then the error is slowly increasing. Thus, the appropriate control action is a positive medium increase in u. If the membership functions for e and $\dot{e}$ are properly defined (typically overlapping by twenty-five percent) and if either or both e and $\dot{e}$ happen to fall into the overlapping area, two or more rules will fire. The controller output du will be an interpolation of the du values for each firing rule. This results in the gradations in the control surface.

Relation between PIs and Fuzzy PIs From the comparison of Figure 1 (a) and Figure 1 (c), it is obvious that a larger number of parameters must be defined to specify the nonlinear surface. As summarized in Figure 2, the linear controller requires only a gain vector, whereas the fuzzy controller needs a rule base, termsets, *and* the equivalent of the gain vector, represented by the input and output scaling factors.

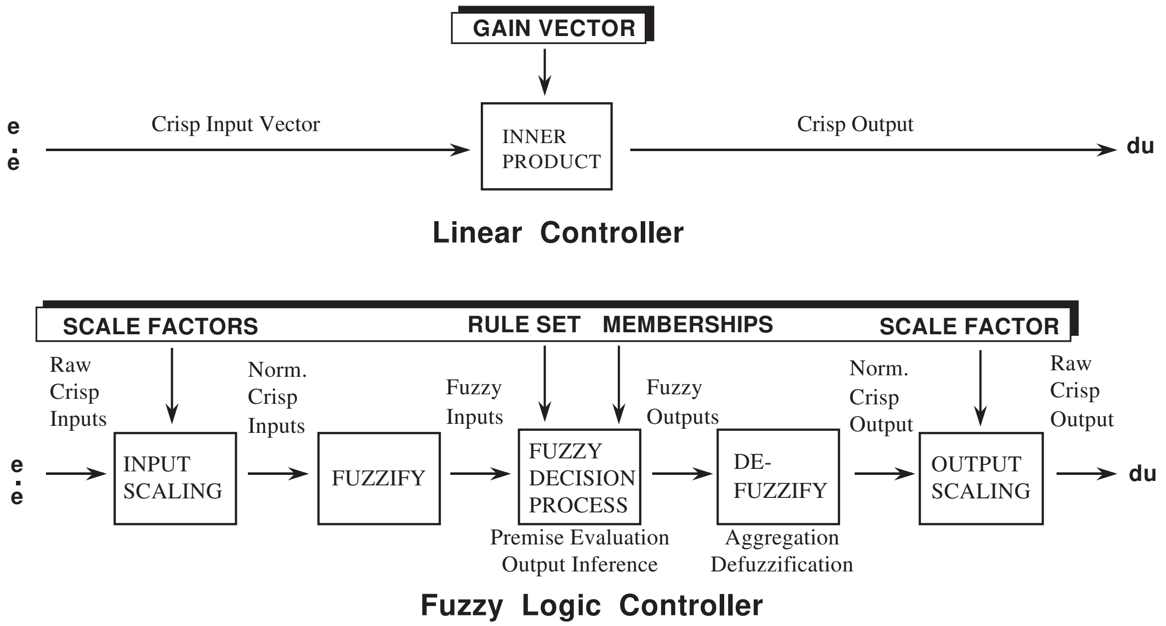

Figure 2: Design parameters for linear and fuzzy PI controllers

This rule set and the associated termsets define the contents of the knowledge base for the fuzzy logic PI. The fuzzy logic analogue of K_p and K_i are reflected in the normalizing factors of the termsets for e and $\dot{e}$. In particular, K_p is approximately equal to $N_{du}/N_{\dot{e}}$, while K_i corresponds to $(N_{du}/N_e)df$, where df is $1/dt$ (Zheng, 1992). By increasing $N_{\dot{e}}$, K_p is decreased. Likewise, by increasing N_e, K_i is decreased (Tang and Mulholland, 1987). Similarly, the normalizing factor of the termset for the incremental control action du is directly proportional to both K_p and K_i.

Since the fuzzy PI is a nonlinear controller, we will extend its comparison to a conventional nonlinear controller, implemented using sliding mode control. In the remainder of this section we will provide an interpretation of the fuzzy logic PI in terms of a two dimensional sliding mode control. A detailed explanation of the sliding mode control and of its generalization to the FLC can be found in references (Slotine and Li, 1991) and (Palmer, 1991), respectively.

2.2.3 Two Dimensional Sliding Mode Controllers

For second order systems, the problem of forcing state vector $\vec{x} = [x\ \dot{x}]$ to track a desired vector $\vec{x_d} = [x_d\ \dot{x_d}]$ can be reduced to the problem of keeping the function

$$s = \dot{e} + \lambda e$$

as close to zero as possible, where e is the tracking error $x - x_d$, $\dot{e}$ is the tracking error derivative $\dot{x} - \dot{x_d}$, and λ is some problem-specific constant.

In two dimensions, the line defined by the equation $s = 0$ is termed the *switching line*. A sliding mode controller attempts to drive the error vector onto the switching line as rapidly as possible, and then force it to the equilibrium point $[e\ \dot{e}] = [0\ 0]$. This is accomplished by defining the control law u as follows:

$$u(s) = \begin{cases} +K & \text{if } s > 0 \\ 0 & \text{if } s = 0 \\ -K & \text{if } s < 0 \end{cases}$$

However, the discontinuity at the switching line $s = 0$ causes extremely high control action if the system does not settle onto the switching line. To remedy this problem, the discontinuity can be smoothed out by a gradation in the region $|s| \leq \Phi$, so the control law becomes:

$$u(s) = \begin{cases} +K & \text{if } s > \Phi \\ +K\frac{s}{\Phi} & \text{if } |s| \leq \Phi \\ -K & \text{if } s < -\Phi \end{cases}$$

The sliding mode controller is linear in the region close to the switching line. For the fuzzy logic PI, the controller output du can be given exponential gains in the region $|s| \leq \Phi$. For instance, if the membership functions for low magnitude du had their centers of mass a distance d away from the origin, the membership functions for medium magnitude could have centers of mass $2d$ away from the origin, and those for high magnitude $4d$ away. This would cause the state vector to approach the switching line faster, reducing rise time.

Settling time can also be reduced by placing a deadband around the switching line close to the equilibrium point. This is done by defining the membership functions for positive low and negative low error so that they stop some small distance away from the point at which the error is zero. The same is done for the error derivative. Thus, when the equilibrium point is approached, there is no change in controller output.

Finally we note that by tuning a fuzzy PI, we can smooth the step-like control surface and we can modify the switching line and generalize it to be a smoother higher-order curve, as illustrated in (Smith and Comer, 1991).

2.3 FLC Technology Development Cycle

The interpreter and the compiler are two of the major elements of the FLC development cycle. The process that leads to the synthesis of a FLC is indeed very similar to that of a Knowledge Based System (Bonissone, 1991b).

2.3.1 The Reasoning Tasks

Three main reasoning tasks are common to KBS and FC: the *knowledge representation*, the *inference mechanism* applicable to the chosen representation and the *control of the inference*. In the next two sections we will briefly cover these tasks for FC. For a more detailed description of KBS reasoning tasks the reader is referred to reference (Bonissone *et al.*, 1987a), while the FC reasoning tasks are extensively covered in reference (Bonissone and Chiang, 1993).

2.3.2 The Application Tasks

As noted in reference (Bonissone, 1990), there are five major stages required to synthesize a Knowledge Based System:

1. requirements and specifications (knowledge acquisition),

2. design choices (KB development),
3. testing and modification (KB functional validation),
4. optimizing storage and response time requirements (KB compilation),
5. running the application (deployment).

The same task decomposition applies to the development of a FLC application:

1. performance function definition, order estimation, state variable identification,
2. KB generation (determination of scaling factors, termsets, and rulesets),
3. stability and robustness analysis, KB tuning,
4. fuzzy rule set compilation,
5. portability/embeddability of the FLC on the target platform.

The first three stages correspond to the FLC development phase, while the last two correspond to its transition from development to deployment.

The use of an interpreter requires the evaluation of *all* the rules in the KB at every iteration. By compiling the KB offline and using a simpler run-time engine, we can reduce the response time and decrease the memory requirements. This feature enables us to implement inexpensive FLCs for cost-sensitive applications. In references (Bonissone and Chiang, 1993, Bonissone, 1991b) the interested reader can find an extensive description of the FLC development and compilation process.

The key advantage of FLCs is their cost-effectiveness in quickly synthesizing nonlinear controllers for dynamic systems. We have reduced design cycle time during the development phase by using an interactive computing environment based on a high level language with its local semantics, interpreter, and compiler. We have achieved efficiency and portability by cross-compiling, the resulting knowledge bases or nonlinear control surfaces prior to deployment.

3 FLC Reasoning Tasks

3.1 The Knowledge Representation

The main representational issues for the FC are: the quantification of the input; the termset definition for each state variable and for each control action; the definition of the type of fuzzy production rule to be used in the KB.

The input quantization consists of describing the input as a fuzzy subset of the input space. This process is necessary to make each input element dimensionally compatible with each state variable. When the universe of discourse of the state is discretized, the FC designer needs to determine a mapping from intervals of the universe of discourse to its point representation. When the universe of discourse is continuous, this mapping is not needed.

The definition of the termset is perhaps the most important of the representational issues. For each state variable and control action, we need to define the granularity of their values (Bonissone and Decker, 1986). Thus, we must determine the cardinality of termsets used to represent each state variable and action, the semantics of the above termsets, and the scaling factors for each state variable and action. These design choices are crucial to issues such as steady-state performance and stability. In particular we have observed that the FC steady-state behavior improves considerably when the termset provides finer granularity around the equilibrium point (Burkhardt and Bonissone, 1992a).

3.1.1 Rule Base

The most common definition of a fuzzy rule base R is the disjunctive interpretation initially proposed by Mamdani (Mamdani and Assilian, 1975) and found in the majority of Fuzzy Controller applications:

$$R = \bigcup_{i=1}^{m} r_i = \bigcup_{i=1}^{m} (\overline{X_i} \rightarrow Y_i) \tag{1}$$

R is composed of a disjunction of m rules. Each rule r_i defines a mapping between a fuzzy state vector $\overline{X_i}$ and a corresponding fuzzy action Y_i. Each rule r_i is represented by the Cartesian product operator. There are, however, other representations for a fuzzy rule, which are based on the material implication operator and the conjunctive interpretation of the rule base (Trillas and Valverde, 1985). For a definition of different rule types, the interested reader should consult references (Mizumoto and Zimmerman, 1982) and (Lee, 1990b). A particularly useful type of FC is the Takagi-Sugeno (TS) controller (Takagi and Sugeno, 1985). In this type of controller the output Y_i of each rule $r_i : \left(\overline{X_i} \rightarrow Y_i\right)$ is no longer a fuzzy subset of the output space but rather a first order polynomial in the state space $\overline{X_i}$, i.e.:

$$Y_i = F_i(\overline{X_i}) = c_0 + \sum_{j=1}^{n} c_j X_{i,j} \tag{2}$$

where $X_{i,j}$ is the jth element of th n-dimensional vector $\overline{X_i}$.

3.2 Inference Engine: Modus Ponens

The inference engine of a FC can be defined as a parallel forward chainer operating on fuzzy production rules. An input vector $\overline{I}$ is matched with each n-dimensional state vector $\overline{X_i}$, i.e., the Left Hand Side (LHS) of rule $r_i = \overline{X_i} \rightarrow Y_i$. The degree of matching λ_i indicates the degree to which the rule output can be applied to the overall FC output. The main inference issues for the FC are: the definition of the fuzzy predicate evaluation, which is usually a possibility measure (Zadeh, 1978); the LHS evaluation, which is typically a triangular norm (Schweizer and Sklar, 1963, Schweizer and Sklar, 1983, Bonissone, 1987); the conclusion detachment, which is normally a triangular norm or a material implication operator; and the rule output aggregation, which is usually a triangular conorm for the disjunctive interpretation of the rule base, or a triangular norm for the conjunctive case.

Under commonly used assumptions we can describe the output of the Fuzzy System as

$$\mu_Y(y) = \bigvee_{i}^{m} \left(min[\lambda_i, \mu_{Y_i}(y)]\right) \tag{3}$$

where λ_i is the the degree of applicability of rule r_i

$$\lambda_i = \bigwedge_{j}^{n} \Pi(X_{i,j} \mid I_j) \tag{4}$$

and $\Pi(X_{i,j} \mid I_j)$ is the possibility measure representing the matching between the reference state variable and the input element[1]

$$\Pi(X_{i,j} \mid I_j) = \bigvee_{x_j} \left(min[\mu_{X_{i,j}}(x_j), \mu_I(x_j)]\right) \tag{6}$$

Equations (3), (4), and (6) describe the generalized modus ponens (Zadeh, 1979), which is the basis for interpreting a fuzzy-rule set.

3.3 Inference Control: Defuzzification

Among the many inference control issues, the most typical are the defuzzification of the FC output, the KB selection for hierarchical or supervisory control mode, the KB self-organizing structure, etc.

The most basic design choice is the selection of the defuzzification mode. The output of the rule aggregation stage is a composite membership distribution defined on the space of control actions. This distribution must be summarized into a scalar value before it is passed to an actuator for execution. This summarization can be performed by a variety of defuzzifiers: the Mean of Maxima (MOM), the Center of Gravity (COG), the Height Method (HM).

[1] When the input is crisp, the degree of matching is the evaluation of the reference membership distribution at the point representing the value of the input:

$$\Pi(X_{i,j} \mid I_j) = min[\mu_{X_{i,j}}(x_0), \mu_{I_j}(x_0)] = \mu_{X_{i,j}}(x_0) \tag{5}$$

3.3.1 Mean Of Maxima (MOM)

The MOM method defines the crisp output y^* as the value of y in which the membership distribution $\mu_Y(y)$ achieves its maximum. If the maximum is obtained in multiple points, y^* is the average of such set of points. Therefore we can define the interval of points $\overline{y^*}$ where such maximum is achieved as

$$\overline{y^*} = \{\hat{y} \in \mathbf{Y} \mid \mu_Y(\hat{y}) = \bigvee_y \mu_Y(y)\} \tag{7}$$

and then we define y^* as the average of $\overline{y^*}$.

3.3.2 Center Of Gravity (COG)

The COG method derives the crisp output y^* as

$$y^* = \frac{\int y \;\; \mu_Y(y) \;\; dy}{\int \mu_Y(y) \;\; dy} \tag{8}$$

3.3.3 Height Method (HM)

The HM method derives the crisp output y^* as

$$y^* = \frac{\sum_{i=1}^{m} \lambda_i \times c_i}{\sum_{i=1}^{m} \lambda_i} \tag{9}$$

where c_i is the center of gravity of $\mu_{Y_i}(y)$.

The selection of the defuzzifier is a tradeoff between storage requirements (MOM lends itself to easy compilation), performance (COG typically exhibits the smoothest performance), and computational time (HM is faster to compute than COG) (Mizumoto, 1989).

4 The Fuzzy KB Compilation Process

The compilation of fuzzy rule bases into fast access lookup tables is analogous to the compilation process used in programming languages and KBS.

Traditionally, the compilation process in programming languages refers to the translation of statements from a high-level source language into a low-level target language, such as assembly language or machine language, to allow for an efficient execution.

The compilation process in Knowledge Based Systems usually refers to the process of translating variables, predicates, and rules into a dependency graph. Such a graph is used to keep a pointer for all rules containing the same variable, and for all variables affected by the same rule, thus eliminating the need of run-time search. The graph typically maintains the current evaluation of each rule, allowing for incremental rule evaluation when new information is entered (Forgy, 1982, Bonissone and Halverson, 1990). Therefore, an interpreter can be used during the KBS development phase, where changing requirements and functionalities may require changes in the source code or in the knowledge base. After a successful validation stage, the KB is considered stable and can then be compiled to minimize storage requirements, avoid exhaustive evaluation, and improve run-time performance.

In a similar fashion, a FC application can be compiled after the validation stage. During the development phase we use the FC interpreter to generate, fine-tune, and validate the fuzzy KB (i.e., scaling factors, term sets and rule set). Once the validation is complete, we employ a FC compiler to generate lookup tables from the fuzzy rule sets.

Let's analyze the FC's two modes of operation, i.e., *interpreted* and *compiled*, by observing the way we synthesize the FC's end-product, i.e. the non-linear control surface. In interpreted mode, at every iterations we fire all the rules to reconstruct the control surface and than we evaluate it for the current input's values. In compiled mode, we first map the current input values to a coordinate set. This set defines the projection on the state space of the *patch* of the control surface that is relevant to the input. Therefore, at every

Counter i $(i = 0 \ldots n)$	Number of non-zero rule outputs $r = 2^i$	Number of cells in M3 pointing to slots containing r rules $\binom{n}{i}(t-1)^i t^{(n-i)}$	Number of cells in slots of M4 pointed to by corresponding cells in M3 $2^i \binom{n}{i}(t-1)^i t^{(n-i)}$
i= 0	1	$t^n = 25$	$t^n = 25$
i= 1	2	$n(t-1)t^{(n-1)} = 40$	$2n(t-1)t^{(n-1)} = 80$
i= 2	4	$\binom{n}{2}(t-1)^2 t^{(n-2)} = 16$	$4\binom{n}{2}(t-1)^2 t^{(n-2)} = 64$
TOTALS		$\lvert M3\rvert = (2t-1)^n = 81$	$\lvert M4\rvert = \sum_{i=0}^{i=n} 2^i \binom{n}{i}(t-1)^i t^{(n-i)} = 169$

Figure 3: Number of elements in M3 for $t = 5$ and $n = 2$.

iteration we construct the relevant patch (by firing a much smaller rule subset) and then we evaluate the patch for the current input's values.

The architecture of a compiled FC, illustrated in Figure 4, consists of four tables (M1, M2, M3, and M4), an address generator, a run-time engine, and a defuzzifier. The interested reader will find more detailed information in reference (Bonissone and Chiang, 1993). Table M1 contains the termsets of the input and output variables. Table M2 lists the rules, with pointers to the termsets that make up the LHS of those rules, as well as pointers to the conclusions of those rules.

The main effort of the compiler is expended in the construction of tables M3 and M4. The state space is partitioned into a number of cells whose boundaries are defined by the termsets of the input variables. If the termsets are overlapping only with adjacent terms, and the number of terms for each of the n state variables is t, then the state space is composed of $(2t-1)^n$ distinct cells, as seen in the last row of Figure 3. In general, the expression for the number of pointers to $r = 2^i$ rules is

$$\binom{n}{i}(t-1)^i t^{(n-i)}$$

where $i = 0 \ldots n$. Each of these cells contains a pointer to a corresponding slot in table M4, which in turn contains a list of pointers to the rules that are applicable in that cell. The maximum length of this list of pointers is 2^n.

In the case of the fuzzy PI, where $n = 2$, M1 contains the termsets for e, $\dot{e}$, and du, while the entire rule set is listed in M2, with the appropriate pointers to the relevant termsets in M1. M3 details a partitioning of the e-$\dot{e}$ plane into a two-dimensional array of size $(2t-1)^2$, with each cell in the array pointing to a slot in M4. Of the $(2t-1)^2$ slots in M4, t^2 contain a pointer to a single rule, $2(t-1)t$ contain pointers to two rules, and $(t-1)^2$ contain pointers to four rules.

At run-time, the values of the input variables and the termsets associated with those variables in M1 are by the address generator to determine into which region of the state space the input falls. Once that region is determined, M3 is consulted, and a list of pointers to the relevant rules in M2 is obtained from M4. The rest of the inferencing process is similar to that of an interpreted FC. However, the run-time process has been made more efficient by avoiding the evaluation of those rules whose contribution is zero. Thus, expression for the Height Method (HM), which was:

$$y^* = \frac{\sum_{i=1}^{m} \lambda_i \times c_i}{\sum_{i=1}^{m} \lambda_i} \tag{10}$$

where c_i is the center of gravity of $\mu_{Y_i}(y)$, is reduced now to:

$$y^* = \frac{\sum_{i|\lambda_i \neq 0}(\lambda_i \times c_i)}{\sum_{i|\lambda_i \neq 0} \lambda_i} \tag{11}$$

where m, the maximum number of rules evaluated in interpreted mode, is a function of the number of state variables n, and the cardinality t of each state variable's termset, namely $m \leq t^n$. After compilation, the

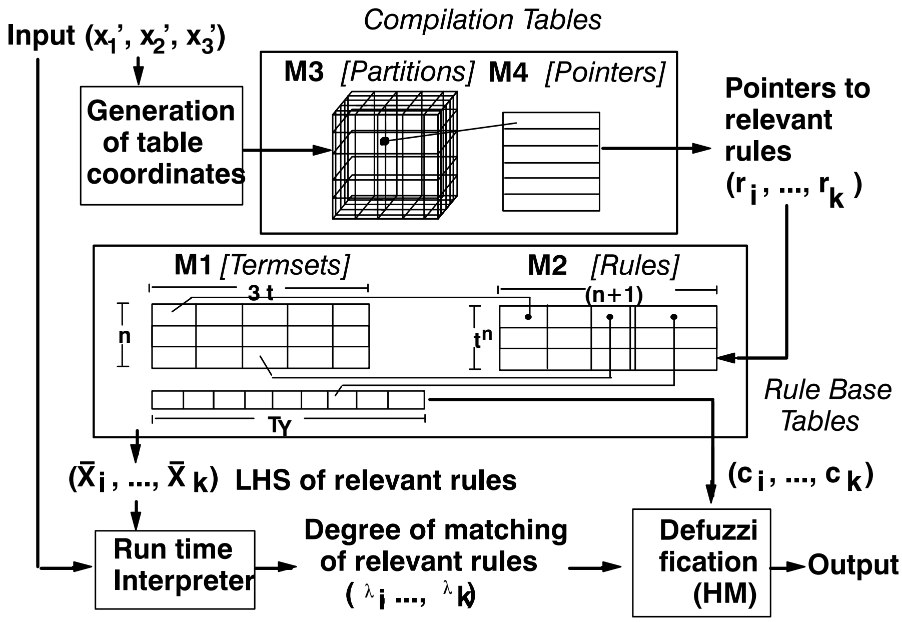

Figure 4: Architecture of a compiled FC, functionally equivalent to that of an interpreted FC.

worst-case maximum number of rule evaluations is $| \lambda_i \neq 0 | \leq 2^n$, but the compiled FC is still functionally equivalent to the interpreted FC. However, the run-time interpreter must still calculate the degrees of matching for the firing rules, perform conclusion detachment, and defuzzify the resulting output.

Figure 5 shows the decrease of run-time rule evaluations obtained by the compilation process. In the worst case, rule execution will be reduced from t^n to 2^n. In the average case, the reduction will be from t^n to about 2-4 rules. The average case was computed as the ratio $\frac{|M4|}{|M3|}$, assuming uniform distributions for the input values and equal partitions size in the state space.[2] The last two columns of the same table illustrate the memory requirements for M3 and M4, i.e., the storage price to be paid for the run-time gains.

Alternatively, to increase the throughput of the controller at the expense of accuracy, run-time rule evaluation can be completely avoided by performing that evaluation during compilation. For each region in the state space partition, a representative point is chosen, and each of the firing rules is evaluated at that point. The resulting output value is stored in M3, instead of storing pointers to lists of rules. At run-time the input values are used to determine the region of state space in which the input falls, and the output value stored there is returned.

This approximation results in a non-uniform sampling of the state space. In the t^n regions where only one rule can fire, the output value is equivalent to the interpreted FC. The control surface of the approximated FC differs the greatest in the $(2t-1)^n - t^n$ regions of overlap, where two or more rules are applicable. In the case of the fuzzy PI, the horizontal areas of the control surface remain unchanged, while the sloping areas

[2] The cells into which the state space has been subdivided have equal size (area, volume, or hypervolume) if the support of any two adjacent inner terms in each termset has a 33% overlapping and the support of any adjacent inner term with any extreme term of every termset has a 50% overlapping. Any smaller overlapping will reduce the size of partitions containing multiple rules and further decrease the average number of run-time rule evaluations.

PARAMETERS		INTERPRETED	COMPILED			
|Terms|	*|State|*	*Rule Evaluation*	*Rule Evaluation*		*Storage*	
t	n	*Total number of Rules:* t^n	*Worst Case:* (2^n)	*Average Case* $\|M4\| / \|M3\|$	*Tot. number of cells* ($\|M3\|$): $(2t-1)^n$	*Tot. number of rule pointers* ($\|M4\|$): $\sum_{i=0}^{i=n} 2^i \binom{n}{i} (t-1)^i t^{(n-i)}$
3	2	9	4	1.96	25	49
3	3	27	8	2.74	125	343
3	4	81	16	3.84	625	2,401
4	2	16	4	2.04	49	100
4	3	64	8	2.92	343	1,000
4	4	256	16	4.16	2,401	10,000
5	2	25	4	2.09	81	169
5	3	125	8	3.01	729	2,197
5	4	625	16	4.35	6,561	28,561

Figure 5: Rule evaluations and storage requirements using compilation.

are replaced with horizontal ones, whose heights are intermediate to those of the boundaries of the sloping areas.

Further details on the compilation process are available in (Bonissone and Chiang, 1993), where we show that compilation can reduce run-time rule evaluations by two orders of magnitude in the *average* case. In the same reference we analyze the memory requirements to show that the compilation can be implemented using the small microprocessors (e.g. 4-8 bit) and limited memory (e.g. 4K-8K bytes) typical of cost-sensitive applications such as appliances and consumer electronics.

5 Cost-Complexity Framework for FC Applications

In reference (Bonissone *et al.*, 1995), we have proposed a cost-complexity framework to show the gamut of FC applications categorized by their cost and performance requirements. This is illustrated in Figure 6. Beside this tradeoff, a third criterion, throughput, is considered a constraint in our comparative analysis. From this analysis, we have observed that as the complexity requirements increase, the structure of the FC becomes hierarchical, following a typical divide-and-conquer strategy. The higher level controller contains the explicit tradeoff knowledge and defines the degree of compromise among the usually competing control goals of the lower level controllers, such as performance versus safety, and fuel efficiency versus emissions.

5.1 Low-Cost/Low-Complexity

The first type of FC applications, exemplified by dishwasher control, is characterized by low cost and complexity. These products are mass-produced and sold at relatively low prices. This type of application usually does not push the performance limits but is constrained by cost. The control of appliances usually consists of simple heuristic decision rules. The typical payoff is smart behavior with low development, deployment, and maintenance costs. Most of these application requirements are addressed by compilation techniques that simplify the path from development to deployment.

5.2 High-Cost/High-Complexity

The second type of FC applications, exemplified by aircraft engine control, is characterized by high cost and high complexity. These products are produced in much smaller quantities but sold at a much higher price. Performance, safety, operational costs, efficiency, and product lifetime are the goals and constraints defining the operational space of the controller. The typical payoff consists of improving performance and efficiency without violating any constraints, including emissions, safety, and components life. Usual approaches addressing these requirements are based on a supervisory control to explicitly account for the various context-dependent tradeoffs. The low-level controllers are implemented by fuzzy or conventional controllers.

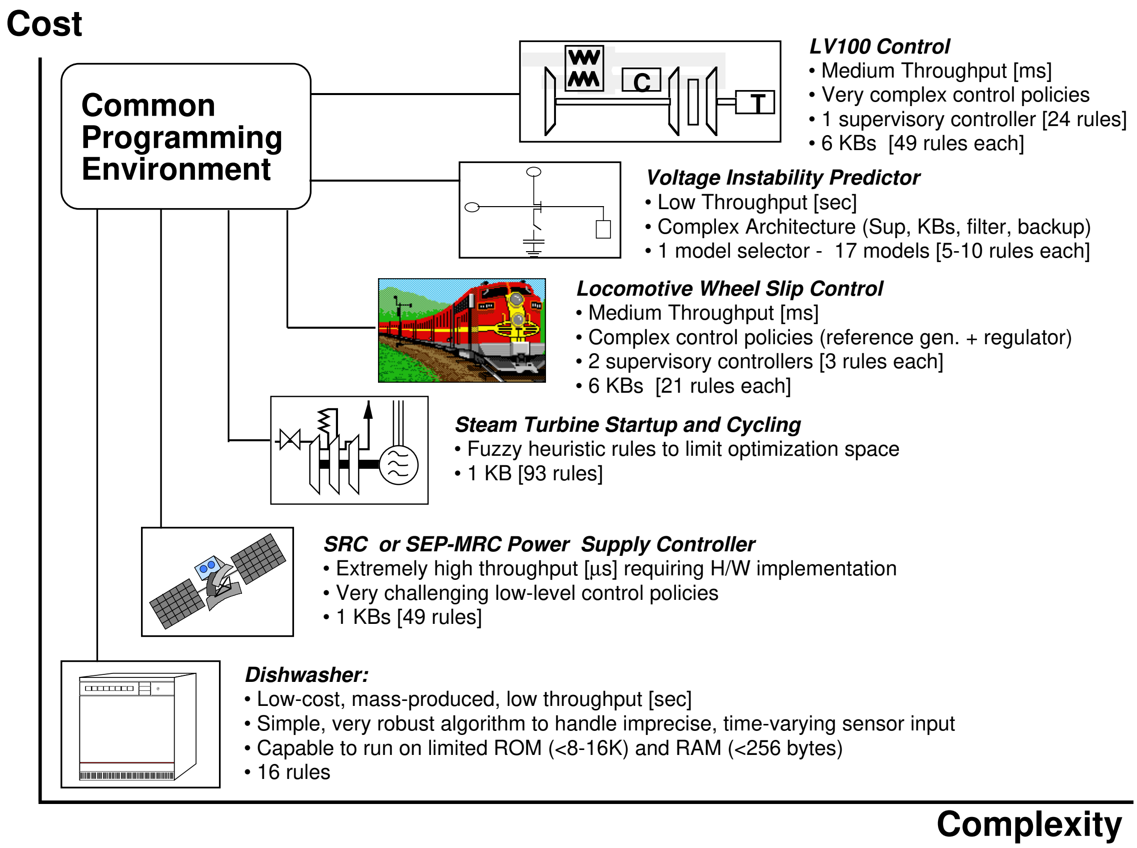

Figure 6: Cost-Complexity Framework for FC Applications

5.3 High Throughput

The third type of FC applications, exemplified by power electronics control, is characterized by extremely stringent throughput requirements. These products typically require response times on the order of microseconds. While cost and functional performance are still significant factors, run-time performance, measured in microseconds, is the critical issue for a successful application. Typical approaches addressing these requirements are based on compilation techniques that approximate the behavior of the controller while avoiding run-time computations. These solutions are then implemented using standard microcontrollers. Alternatively, special purpose hardware realize the fuzzy controller.

5.4 Hierarchical Control

Orthogonal to the above dimensions (cost, throughput, and performance) is the concept of hierarchical control. A hierarchical control scheme permits the decomposition of complex problems into a series of smaller and simpler ones. As these simpler problems are solved, typically by using low level controllers, they can be recombined to address the larger problem. This recombination is governed by a fuzzy logic supervisory controller that performs soft switching between different modes of operation. The soft switching allows more than one mode of operation (with its corresponding controller) to be active at any one time. By assigning a linear combination of low level controllers to a given mode, the engineer can trade off safety and efficiency against performance.

Another important feature of fuzzy hierarchical control is its inherently simpler software maintenance. By explicitly representing in the supervisory rule set our policy for managing potentially conflicting goals (lower level controllers), we are defining our willingness to tradeoff performance under one criterion, such as speed, for better compliance under a different criterion, such as safety. These tradeoff policies are context dependent, as they are conditioned by the left-hand side of the supervisory rules. Therefore, different supervisory modes entailing different tradeoff policies, for instance the compromise between tank speed/acceleration and engine lifespan in training mode or in battle mode, can be achieved simply by selecting different rule bases for the same supervisory controller. We will illustrate our ideas by describing two fuzzy supervisory control applications to engine control and steam turbine startup.

6 PART I Conclusions: FLC Technology

We have implemented Fuzzy Logic Controllers using a high level language with its local semantics, interpreter, and compiler.

By using an interactive computing environment based on a FC interpreter, we have considerably shortened the design cycle of the above applications, while achieving comparable or better performance than purely conventional control. After validating the synthesized nonlinear control surface, we have improved portability and decreased deployment cost by compiling the resulting fuzzy KB and by using efficient run-time engines.

The required design cycle time could be reduced even further if we could avoid most or all of the manual tuning procedures currently used in the last part of the FC development. One of our current research efforts is indeed aimed at extending adaptation techniques developed in other fields to provide automatic tuning of FCs. These techniques typically take the existing rule network defined in the FC knowledge base and provide on-line adaptation for the rule parameters. These synergistic combinations will be further discussed in Part III as part of the broader field of Soft Computing.

Having provided a cost-complexity framework for FC applications, we will now focus our attention to an illustrative sample of such applications.

7 PART II: Sample of Industrial Applications

In this section we will briefly describe our experience in developing and applying fuzzy logic technology to a variety of control problems, including applications in fuzzy hierarchical control, hybrid control, and high throughput control.

7.1 Recuperative Turboshaft Engine Control (LV100)

7.1.1 Problem Description

The LV100 is a recuperative turboshaft engine, with a high pressure spool supplying an airflow to a free power turbine, in order to generate the high torque necessary for moving a heavy vehicle from rest. Gas turbines have higher fuel consumption rates than diesel engines, the alternative power sources for heavy vehicles. To address the fuel consumption problem, both a heat exchanger (recuperator) and a variable area turbine nozzle (VATN) are included to increase the efficiency of the thermodynamic cycle (Bonissone and Chiang, 1995). A schematic of this engine is illustrated in Figure 7.

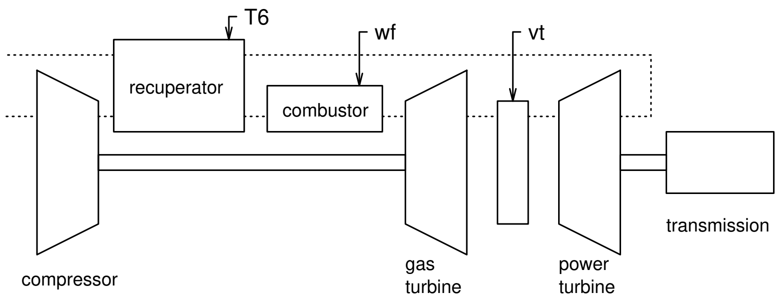

Figure 7: Schematic of the LV100 engine.

Current Solution. In the baseline control version of this system, the engine is controlled by ten low level controllers, designed to govern the engine when specific conditions, or modes, are sensed. Each time step, a dominant controller is selected from that group by a network of minimum and maximum functions and its output passed on to the actuators. Because only one controller is dominant at a given time, abrupt changes in control action may occur as regulation of the engine passes from one controller to another.

7.1.2 Solution Description

Hybrid Control System: Fuzzy Supervisory and Conventional Low level Controllers. We first identified twenty-four individual operating modes, and replaced the min/max network (the crisp mode selector) with a FLC selector that identified modes and utilized the interpolating action of fuzzy logic to blend the outputs of two or more controllers. The hybrid system control performed better than the baseline control, but still exhibited some limitations (see section 7.1.3.)

Fuzzy Control System: Fuzzy Supervisory and Fuzzy Low level Controllers. Therefore, we replaced the conventional low level controllers with fuzzy logic PI ones. Testing of the controls was performed on a component level simulation of the gas turbine and its associated transmission. Efficiency was improved, achieving nontrivial fuel savings.

This hierarchical controller architecture is similar to one used for the control of a small helicopter (Sugeno *et al.*, 1991). In the helicopter control, three low level controllers are used for roll, pitch, and yaw stability, while a fuzzy logic supervisory control processes pilot commands and perturbs the inputs to the stability controllers. In the gas turbine power plant control, six low level controllers are used to govern fuel flow and turbine nozzle area, while a fuzzy logic supervisory control processes driver commands and combines the outputs of the low level controllers.

Advantages of the FLC System. Using fuzzy logic to implement a mode selector provides a number of advantages:

- Instead of having only a single controller active at a given time, modes can be defined with many controllers active simultaneously. When switching is performed between two or more modes, the inferencing method of fuzzy logic interpolates between the control actions for those modes, resulting in smoother transitions.
- The action of the mode selector is more readily apparent because the dominant mode or modes can be found by examining a set of engine parameters, whereas the min/max ladder requires the low level controllers to output excessive rates so that those controllers will not be selected.
- Fuzzy logic mode selection allows the tradeoffs to be moved up into the mode selector, so that the low level controllers handle the dynamics, while the higher level mode selector deals with quasi-steady state conditions, thus making the design of low level controllers easier.

7.1.3 Results, Analysis, and Observations

The results obtained from the fuzzy control system can be described using many criteria, such as performance, fuel consumption, component life expectancy, and cost.

Performance. The response of the plant has been improved with respect to the conventional control scheme. Shorter rise times (due to the non-linear gain of the controllers) were obtained in conjunction with reduced overshoots and faster settling times (due to highly tuned low level PIs).

Fuel Consumption. Fuel consumption was considerably lower than the one obtained by using conventional control schemes. The fuzzy logic mode selector was able to run the engine must closer to its optimal temperature, resulting in a more efficient thermodynamic cycle. Further fuel savings were realized by replacing the conventional controllers with fuzzy logic PIs and decoupling the actuator actions.

Component Life. The fuzzy control system provided excellent performance and fuel savings without sacrificing component life. All operational limits for gas and power turbines, transmission, and recuperator were maintained.

The fuel consumption and component life results are best illustrated in Figure 8, which shows three plots. In each plot, the *solid* line represents the maximum T_6, the temperature which should not be exceeded by the exhaust gases to preserve the normal life of the heat exchanger (recuperator) illustrated in Figure 7. The *long-dashed* line shows the desired T_6, the optimal temperature which would guarantee minimum fuel consumption. These two references change as a function of the load and are computed on-line by a thermodynamic model that runs with the controller. Finally, the *short-dashed* line shows the actual $T_6$of the exhaust gases as they enter the recuperator.

In Figure 8(a) we can observe the results obtained from the the crisp mode selector and the conventional low level controllers. The control system tries to follow the desired T_6, but we can observe that the gap between desired and actual T_6, representing fuel inefficiency, is significant. Furthermore, the actual temperature T_6 hits the maximum T_6 in two places (at t = 15 sec. and t = 470 sec.), as the load (not shown) suddenly changes. This excursion clearly impacts the recuperator's life expectancy.

Figure 8(b) displays the results from the hybrid control system. We can observe a considerable improvement in fuel consumption, evidenced by a reduction of the gap between desired and actual T_6. However, the actual temperature still hits the maximum temperature in the same two locations. Finally, Figure 8(c) shows the results from the fuzzy control system. The gap has been drastically reduced, leading to considerable fuel savings. Furthermore, the actual temperature T_6 never reaches the maximum T_6, extending the life of the recuperator.

Development Cost. As a final remark, we would like to note that the total effort required to design, develop, test and tune the FLC system was slightly less than 25% of the time required to develop the baseline control system. However, the existing baseline controller gave us the additional benefit of an improved qualitative understanding of the plant's behavior.

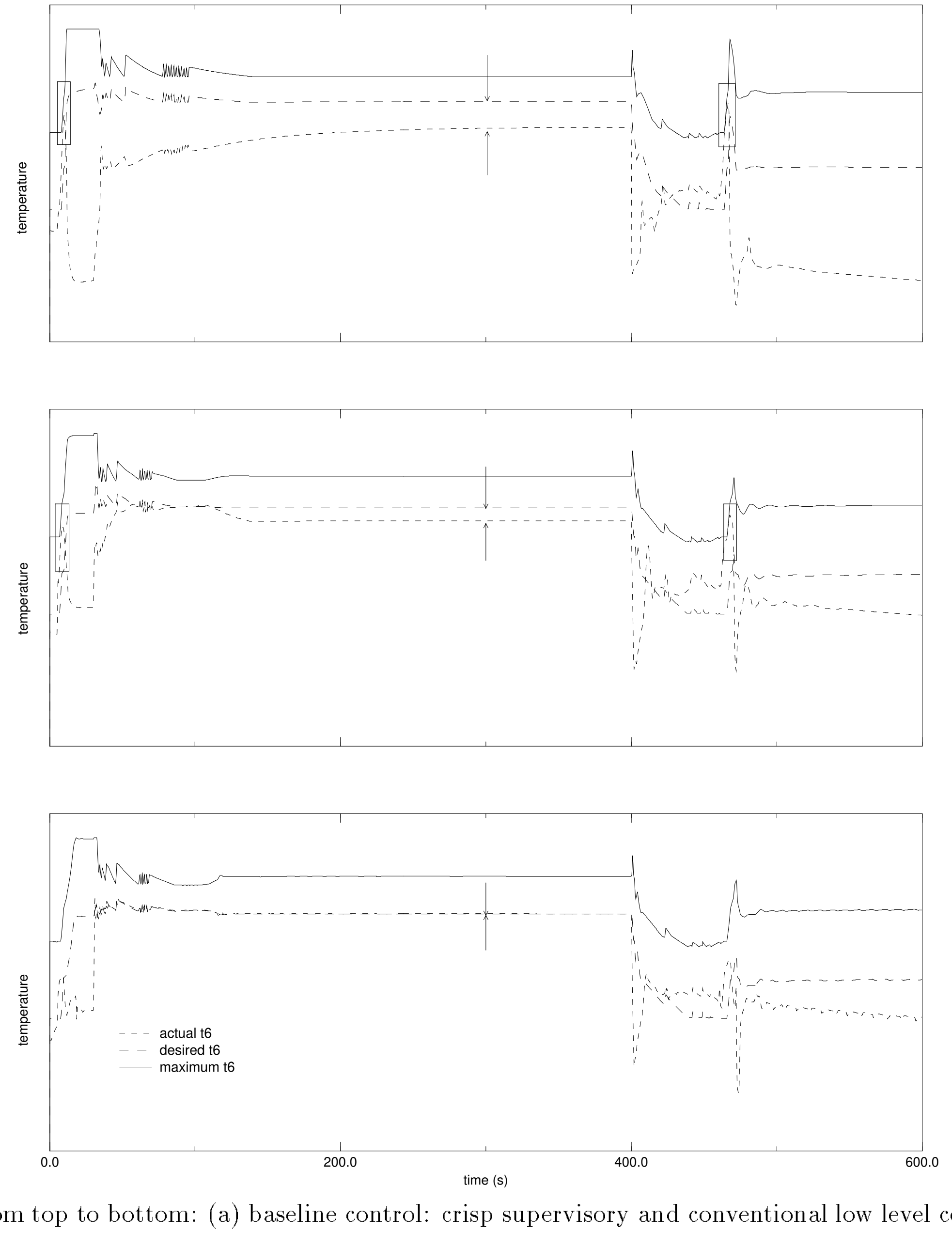

Figure 8: From top to bottom: (a) baseline control: crisp supervisory and conventional low level controllers, (b) hybrid control: fuzzy supervisory and conventional low level controllers, (c) fuzzy control: fuzzy supervisory and fuzzy low level controllers.

7.1.4 Application Conclusions

On the whole, fuzzy logic and fuzzy logic mode selection is experimentally tractable, providing performance comparable to that of the conventional control scheme. In the design of a conventional PI controller, there are a number of parameters available for adjustment, namely the integrator time constant, and the input and output gain vectors. The designer of a typical fuzzy logic PI controller has to determine, in addition to input and output scaling factors, the desired membership functions and rules. These additional degrees of freedom allow the controller to achieve better performance at the expense of more complex tuning procedures. This is currently one of the active research topics in FLC technology. In summary, the reduced development cost, combined with the fuel savings, improved performance, and component life preservation, show the tremendous potential of this technology in addressing complex, hierarchical control problems. This is further illustrated in the next application.

7.2 Steam Turbine Start-Up

7.2.1 Problem Description

The process of warming a large steam turbine after an extended period of maintenance or inactivity is called *prewarming*. Currently, many large steam turbines are prewarmed manually in an otherwise completely automated startup procedure. An approach is outlined for automating the prewarming of large steam turbines using fuzzy logic. In this approach, heuristics employed by a human operator during prewarming are captured in a fuzzy rule base and used by a supervisory controller.

In rotor prewarming, the goal is to bring the temperatures of critical rotor locations to specified values before the turbine is accelerated to nominal operating speed. The reason for rotor prewarming is to minimize the stresses induced in the metal when exposed to main steam temperature and pressure conditions. During prewarming, the turbine is rotated at a constant low speed of 2.43 rpm. Any acceleration of the turbine beyond this speed is referred to as a *roll-off*. The frequency of roll-offs has to be minimized during the prewarming procedure, as it is time consuming to bring the turbine back onto turning gear. The main control objective during prewarming is to minimize the time to reach the desired turbine component temperatures while satisfying the following constraints: (i) limit steam flow so as to prevent frequent roll-offs, (ii) limit temperature gradients in the high-pressure turbine rotor and valve chest so as to stay within specified stress margins, and (iii) minimize the buildup of condensate in the turbine shell that could lead to cavitation of the rotor buckets.

The rotor prewarming problem lends itself to a rule-based control approach because no good models are available for turbines at low temperature and pressure conditions. The lack of appropriate models is why traditional control oriented approaches to automation of this process have not been successful to date. However, human operators routinely perform prewarming. This is the motivation for using fuzzy logic, which can capture heuristic knowledge in a rule base (Lee, 1990a). Fuzzy logic has been used previously in steam turbine startup, but in the stages following prewarming (et al., 1988).

The benefits of automating this process are (i) extending the life of the turbine, (ii) enabling the turbine to be brought to roll-off conditions faster, and (iii) creating uniformity in the prewarming procedure across installations, and within an installation, for different operators.

Current Prewarming Procedure. The present approach to prewarming of turbines consists of two phases In condensation control, the metal is heated to a temperature above the condensation point by admitting steam through a valve, the main steam valve bypass valve (MSVBV). Drain valves are wide open during this stage to allow the condensate to drain, DV(2-5). In the second phase, pressure regulation, the goal is to pressurize the high-pressure (HP) and reheat (RHT) turbines to a specified value by admitting steam slowly. During this phase the drain valves are kept almost completely closed, but sufficiently open to allow the remaining condensate to escape. The rotor is then allowed to *soak*, while continuing to allow steam in through the MSVBV to bring the rotor temperature and pressure to rated values.

The process described above is a manual one at the present time. The goal of this work is to automate the process, while providing the operator with a *knob* to control the prewarming rate.

7.2.2 Solution Description

Prewarming Automation Procedure. The prewarming automation approach is based on a combination of conventional closed-loop control and fuzzy logic supervisory control.

Several low level regulatory loops are used in the controllers for steam turbines today, specifically in controlling temperature, speed, stress, and drain valve. Such control loops are traditionally implemented using conventional control, such as PIs. There are two primary control actuators, the MSVBV and the DVs. Opening the MSVBV allows more steam into the turbine, causing a faster rise in temperature and pressure. The DVs have a secondary effect in that allowing condensate to escape will also result in steam escaping, thus increasing the warmup time. The three controllers output incremental bypass valve commands (DMSVBV). The drain valve control loops generate an incremental control action (DDV) to the drain valve.

Fuzzy Logic Supervisory Control. Each of the regulatory loops described above is responsible for maintaining a particular process variable at a specified set point. Some of these objectives may be in conflict when an overall control objective for the system is specified. For example, if the top level control objective is to prewarm the turbine as fast as possible, then the low level control objectives of draining condensate, preventing excessive stress in the rotor bore, and keeping the rotor on turning gear have to have a lower priority than temperature control. Similarly, when the top-level control objective is to prewarm the turbine at a slower rate, the condensate, stress, and speed control loops have a higher priority than the temperature regulation loop. The goal of the supervisory controller is to manage the priorities of the lower level control loops, given a higher level control objective by an operator.

In this approach, the operator is provided with a single input that allows specification of the prewarming rate in a range from *slow* to *fast*. Depending on the prewarming rate selected by the operator, and the current system state (temperature, speed, stress, and condensate), the low-level control loops are weighted to varying extents by the fuzzy mode selector to produce control actions for the bypass valve (DMSVBV) and drain valves (DDV). The weighting of the individual regulatory loops is determined by a set of fuzzy rules. Currently eight rules are used to implement the supervisory control function; variations on these rules could be used to produce different system behaviors, if desired. The inputs to the rules are the operator-specified warmup rate, the speed error, and the stress error. Other inputs could also be incorporated, such as pressures and operator observations. The outputs of the fuzzy controller are the weights assigned to the each of the regulatory control loops corresponding to speed, stress, temperature, and drain valves.

7.2.3 Results, Analysis, and Observations

Figures 9 and 10 illustrate the operation of the supervisory controller in simulation. In each figure the system variables of interest are rotor bore and surface temperatures, rotor speed, and the condensate accumulation in each of four sections of the turbine. In Figure 9, the operator has selected a *slow* warmup rate. The effect of this selection is evident in the time plots of the system variables. The rotor is almost always on turning gear, the condensate accumulation is almost zero, the rotor bore stress is well below the allowable limit of 6500 psi, and the time taken to reach the desired rotor bore temperatures of 300 F is long (12 hours in simulation). Figure 10 shows the effect of the operator selecting a *fast* prewarming rate. The rotor is off turning gear speed much longer, the condensate accumulations are much greater, and the rotor bore stress exceeds the allowable limit for a longer time, but the prewarming time is much shorter (4 hours). These examples of system operation illustrate only the effect of operator input to the fuzzy supervisor. In addition, the system states of speed and stress also influence the weight assignments for the low level control loops.

7.2.4 Application Conclusions

This successful application of fuzzy supervisory control to the startup problem illustrates the leverage of combining plant operational requirements and operator expertise in one explicit fuzzy rule base.

Operator observations can be blended with existing fuzzy control rules to augment information not available from sensors. For example, condensation rate observations, inferred by inspecting drain valve conditions, will be necessary in cases where the drain valves are not instrumented with temperature sensors.

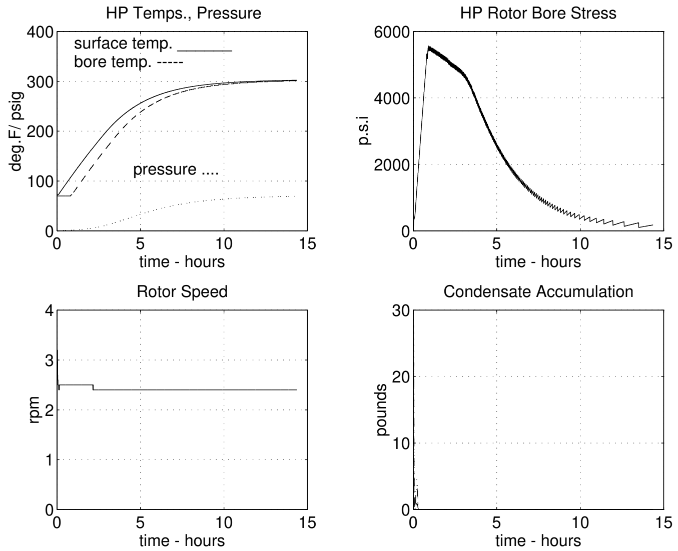

Figure 9: System Performance for Slow Warmup Rate

8 Hybrid Control Applications

Beside its natural role as the language to express tradeoffs at the supervisory level, fuzzy logic has proven to be an expressive control synthesis tool at lower levels. We have experienced large payoffs in combining FLC technology with other control techniques such as conventional optimal control and neural network inferencing.

We will use two examples to illustrate this synergy. The first is the turbine cycling control problem in which FLC is combined with optimal control (see reference (Marcelle *et al.*, 1994) for a more detailed explanation). The second is the use of a fuzzy neural system to extract models from very sparse field data.

8.1 Conventional and Fuzzy Control: Optimal Load Cycling of Large Steam Turbines

8.1.1 Problem Description

Increasing competition from non-utility power generators, growing electricity demand, and rising costs of generation equipment have induced utilities to operate existing generation plant and equipment in the most flexible and cost-effective manner possible (Armor *et al.*, 1985). The impact of these market forces has been particularly significant on the operation of steam power plants.

Electric load demand varies in a periodic manner and hence it is very predictable. Well known factors such as the time of the day, the day of the week, and the month of the year are directly correlated to the load demand. These and other factors, along with historical data, are used by utilities to generate accurate projected load profiles(Khotanzad *et al.*, February 1993). The minimum megawatt demand of the projected load profile is called the base load; any demand above base load is termed peaking load. It is common for peaking loads to be double the base load value at certain times in the projected load profile. To meet peaking load demand, selected power plants are used to follow the large percentage load change ramps, while the base load is supplied by plants running continuously at rated megawatt output.

Large steam power plants are increasingly required to perform double duty as both base load and peaking units. The availability of less expensive base load power from independent producers and nuclear plants has forced utilities to cycle their fossil power plants to economically meet peak demand. The trend toward deeper and more frequent cycling of fossil units, in conjunction with the need to meet increasingly stringent emissions and efficiency targets, has rejuvenated interest in advanced control methods as a means to satisfy these demanding performance requirements.

Cycling of steam turbines induces large thermal stresses in the thick metal parts of both the boiler and turbine as a result of steep steam-to-metal temperature gradients. Turbine rotor stress is the most critical of

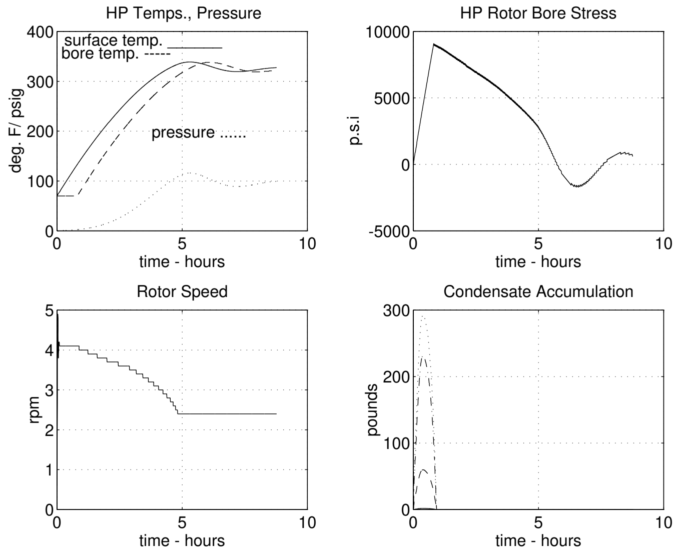

Figure 10: System Performance for Fast Warmup Rate

these stresses and limits load ramping rates. Elevated stress levels reduce equipment service life and increase maintenance costs.

The control objective in power plant operation is to meet demand at minimal cost with available generation equipment. An effective control strategy must be able to directly assess the cost of different modes of operation which can satisfy that power demand. The best control strategy satisfies the demand with the lowest cost.

Current Solutions. There are two commonly used control modes for steam turbine operation: fixed and sliding boiler pressure control(Hanzalek and Ipsen, November 7 11 1965). Under fixed boiler pressure control, the turbine governor valve is used to control megawatt output while boiler pressure is kept nominally constant. Fixed pressure control allows rapid load changes to be readily followed at the expense of large thermal stresses. These stresses are induced in the turbine as a result of inlet steam temperature fluctuation.

Under sliding pressure control, boiler pressure is used to control megawatt output while the turbine governor valve is kept wide open. Sliding pressure control causes minimal thermally induced stress, at the expense of slow megawatt response. This response is slow because large boiler inertias must be overcome to alter saturation conditions in order to ramp up pressure.

8.1.2 Solution Description

To produce good load tracking and minimum thermal stress it is desirable to simultaneously use both methods of operation. The control strategy described in this section minimizes cumulative turbine stress and load tracking error by coordinated control of both the turbine governor valve and boiler pressure. This is accomplished while observing practical system constraints, such as valve rate limits and maximum operating pressure.

Architecture Type. The controller architecture is illustrated in Figure 11. There are four key ideas underlying the design of the controller:

- Future load demand from projected load profiles, which are accurate and readily available information at power plants, is considered when deciding on the current control action. Optimal future performance is therefore not jeopardized by focusing only on short term demand, and the possibility that current modes of operation will result in future constraint violation is greatly reduced.
- Fuzzy logic is utilized to encode operator experience and intuition in establishing priority among conflicting performance objectives. For example, load tracking can be traded off against stress minimization or turbine valve throttling against varying boiler pressure. This intuition provides long term

guidance for overall cycling cost optimization, and is transferred to the model predictive controller (MPC) through weights in a cost function.

- Model predictive control is used to optimize the tradeoff between good load following and equipment stress minimization during cycling operations (Martin and G. J. Silvestri, May 10 12 1988). The MPC algorithm finds the optimum valve/pressure setpoint trajectory which minimizes the cycling costs over the short term. Long term overall performance is not sacrificed however, since the weights in the cost function are chosen based on long term objectives.
- Hard system constraints such as maximum safe operating pressure or valve closure rate are explicitly included in the algorithm and never violated. Soft constraints such as desired pressure at the end of a load change are also included. These soft constraints and the priority given to satisfying them is determined by fuzzy logic.

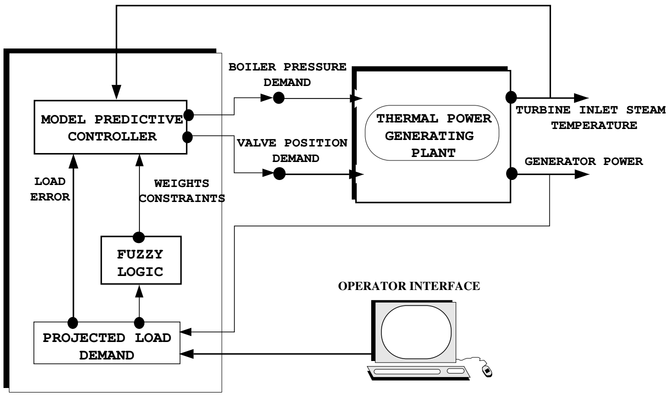

Figure 11: Hybrid Fuzzy MPC Controller Architecture.

Model Predictive Control Algorithm. Model predictive control (MPC) has been successfully implemented in several process and power plant control applications since the mid 1970's. MPC is a method of utilizing efficient optimization algorithms to solve complex control problems with critical input, output and/or state constraints.

At the core of this control methodology is the assumption of availability of a high fidelity model of the plant, process, or system to be controlled. In addition, the desired system trajectory is required. With this system information, the optimization algorithm generates a sequence of input or controller commands which simultaneously achieves the desired system trajectory and minimizes the cost of maintaining the plant on that trajectory.

In this application, the plant model is obtained from a simulation using PC Trax, the power industry standard simulation environment. The desired system trajectory is the projected load demand.

The model predictive control technique used for the optimal control algorithm is quadratic dynamic matrix control (QDMC), of which a detailed presentation can be found in (Garcia and Morshedi, 1986). The essence of the algorithm is the minimization of the cost function J. This minimization can be expressed by the quadratic program (QP)

$$\min_{x(k)} J = \min_{x(k)} \{\frac{1}{2}x(k)^T Hx(k) - g(k+1)x(k)\}$$

Solving the QP at each time k produces a vector sequence $x(k)$, which are the optimal controller setpoints for tracking the desired trajectory at minimal cost. The QP matrix H contains the plant model A, the actuator weighting matrix Λ, and plant output weighting matrix Γ. H is given by

$$H = A^T \Gamma^T \Gamma A + \Lambda^T \Lambda$$

$g(k+1)$ is the QP gradient vector $A^T \Gamma^T \Gamma e(k+1)$, where the vector $e(k+1)$ is the error from the desired system trajectory.

Weights in Γ correspond to the system trajectory tracking errors in $e(k+1)$ and the relative magnitude of these weights determines the priority the QDMC algorithm places on reducing particular errors to zero. Each weight in the matrix Λ corresponds to a system actuator in $x(k)$. The relative magnitude of these weights determines the degree to which each actuator is used in minimizing the errors in $e(k+1)$. These cost function weights are provided by the fuzzy logic knowledge base.

Fuzzy Logic Knowledge Base. The fuzzy logic knowledge base provides a ranking of system performance priorities from a projected load profile. A load profile consists of alternating constant load stages and ramping load stages. The profile is decomposed into groups of sequential stages, as in Figure 12.

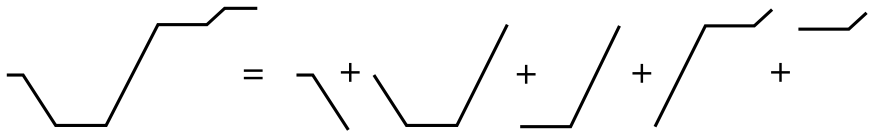

Figure 12: The decomposition of a typical load profile.

There are two types of groups: those that begin with a ramping load stage and those that begin with a constant load stage.

The ramping load groups consist of three load stages, beginning and ending with ramping load stages. They are characterized by initial loading rate, initial load change, constant load duration, subsequent loading rate, and subsequent load change, as in Figure 13(a).

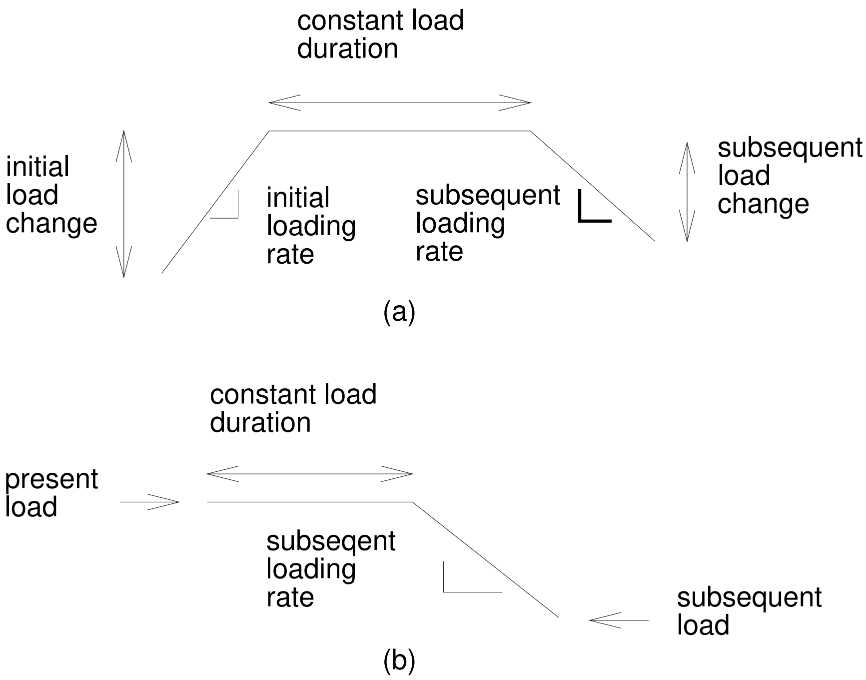

Figure 13: Characterization of: (a) ramping load group, (b) constant load group.

The constant load groups consist of two load stages, beginning with a constant load stage and ending with a ramping load stage. They are characterized by the present load, constant load duration, subsequent loading rate, and subsequent load, as in Figure 13(b).

The ramping load groups allow the controller to respond to immediate cycling demand, while preparing for subsequent cycling. The constant load groups are used as a means of bridging the gap between the current system state and the desired state for executing the subsequent cycling operation.

Membership functions for the features of these groups have been defined; constant load lengths are either short or long, loading rates are slow or fast, load changes are small or large, and loads are low, medium, or large.

These features have been coded into sets of rules which select cost function weights to minimize load tracking error and turbine inlet temperature changes and to prioritize pressure and valve position changes. The weights can be low, medium, or high. Large weights imply that small changes have large attendant costs; changes in these variables are then suppressed by the QDMC optimization algorithm. For instance, low weights on temperature changes mean that large temperature changes are permitted.

Load tracking error is always of high priority. Therefore it has been assigned a high weight for every situation. Other cost function parameters can be assigned equal or lower priority by the fuzzy logic knowledge base.

In addition, the rules determine target pressures to be achieved at end of constant load groups. These targets are soft operating constraints, and the priority assigned to meeting them is also in the form of a fuzzy weight.

To illustrate the operation of the fuzzy logic, consider a typical load cycling scenario illustrated in Figure 14.

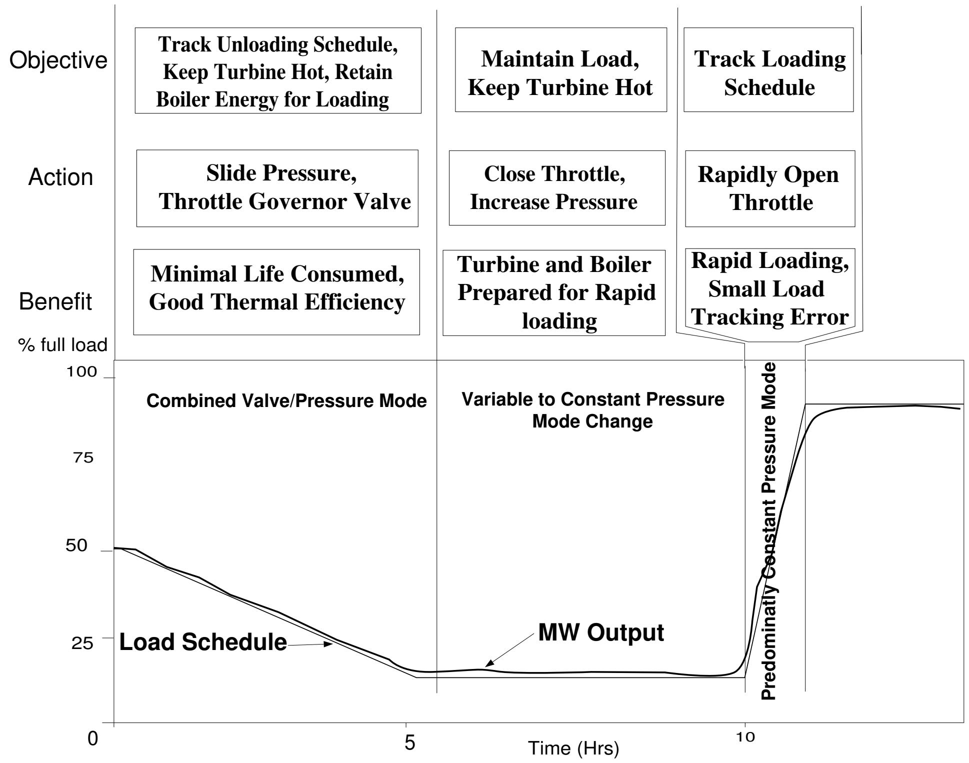

Figure 14: Optimal multi-objective turbine metal temperature and load following control over time horizon.

The load profile consists of a gradual unloading with a medium load change, a constant load stage of short duration, a fast reloading with a large load change, and a long constant load stage thereafter.

A heuristic strategy to satisfy the control objectives given the load profile in Figure 14 is to: unload using a combination of sliding pressure and valve throttling to keep the turbine temperature high, induce minimal levels of stress, provide good load following, and retain boiler potential energy for pending rapid loading; gradually increase pressure and throttle turbine governor valve during the short constant load stage to boost boiler potential energy in preparation for the rapid loading; and finally rapidly load by opening the governor valve to meet the desired load demand. By coordinating boiler pressure and governor valve position, the governor valve can be kept fairly wide open. Turbine inlet steam temperature changes are then kept small, minimizing thermal stress.

To show the role of the fuzzy logic knowledge base, reconsider the example given above. At the beginning of each stage in the load profile, the knowledge base is consulted to provide the MPC with a set of weights for its cost function. These weights govern how the MPC executes the cycling stage. The knowledge base also produces target pressures when constant load groups are encountered.

The ramping load and constant load groups are further subdivided to account for the sign of the load ramps in the ramping load stages. This refinement of the knowledge base results in approximately one hundred rules that cover all possible load profiles.

8.1.3 Results and Analysis

The performance of the fuzzy-MPC controller is compared to that of both a fixed and a variable pressure controller to demonstrate the derived benefits from coordinating these two modes during load cycling, trading off stress against load tracking. In situations where load cycling ramps are steep, variable pressure operation causes poor load tracking and constant pressure operation would induce large thermal stresses, the fuzzy-MPC controller utilizes advance knowledge of the load profile to prepare for the turbine for the transition. This minimizes stress while maintaining good load tracking.

The fuzzy-MPC controller is also compared with a standard MPC controller with fixed weighting matrices and the same ten minute MPC horizon, highlighting the importance of the long term prediction of the fuzzy logic in selecting weights for the current cycling operation. In situations where long constant loads are followed by large load increases, which can occur during early morning hours, inappropriate short term minimization of stress results in extremely poor long term load tracking performance as a result of ignoring future demand.

Finally the fuzzy-MPC controller with a ten minute MPC horizon is compared to a fuzzy-MPC controller with a two hour MPC horizon. By extending the MPC horizon, local performance is sacrificed with the two hour horizon because the optimization algorithm equally weights performance two hours into the future with performance at the next time step. The resulting controller setpoints for the next time step become imprecise, because they are not tightly focussed on locally optimizing performance.

In short, the short horizon controller in contrast has effectively separated what are indeed separate issues. The long term anticipative features provided by the fuzzy logic prevents myopic local optimization which can lead to poor future performance. At the same time excellent local load following and stress minimization is produced by the MPC algorithm with a ten minute predictive horizon.

8.1.4 Application Conclusions

A novel hierarchical coordinated control scheme for the cycling of steam turbine¡s has been described. The scheme consists of: a projected load demand profile, a model predictive control algorithm, and a higher level fuzzy logic knowledge base. The load demand profile provides a desired trajectory for plant megawatt output. The model predictive control algorithm determines the optimal actuator commands to track the desired load profile and minimize the cost of power generation. The fuzzy logic knowledge base provides long term optimization guidance by selecting present operational priorities based on present and future cycling demand.

This hybrid controller produces good load tracking performance, as well as good long term operation cost minimization. By utilizing fuzzy logic to scan over an extended horizon, the computational requirements for the optimization algorithm are reduced without sacrificing long term performance. The technique can be easily extended to include other performance metrics, such as fuel consumption and cycle life expenditure limits. This hybrid approach is a generic technique for simultaneously achieving short term and long term control objectives. It can be applied to any process, provided desired long term trajectories and heuristic control insights exist. A more detailed explanation of this application can be found in reference (Marcelle *et al.*, 1994).

8.2 Neural and Fuzzy Systems: Oil Film Compensation in Steel Mill

8.2.1 Problem Description

In steel rolling mills, rollers exert force on a moving sheet of steel to control its thickness. The oil film present on the roller bearings introduces a distortion which is sufficient to violate the tolerance limit on steel thickness. To compensate for this, a correction factor has to be applied to the process. This correction (Δs) depends on a number of factors, some of which are the roll's rotational speed (ω), the force (F) exerted by the stand, and roll acceleration. A mapping is required that will determine Δs from the two most relevant physical variables: ω and F.

Since there is no analytical model of the process, all approaches have to make use of a limited set of data points which specify the appropriate correction for some specific settings of the input variables ω and F. Each such data point is collected experimentally, and at considerable cost. Only 64 data points (triplets of values) are available, from which the input-output relationship $(\omega, F) \mapsto \Delta s$ must be inferred.

In addition to modeling the mapping, it is desirable to *reverse engineer* the relationship by extracting rules from the data. This helps in understanding the relationship, so that the same knowledge is used for different rolls in different configurations. This is important, as collecting data for each roller and stand is expensive. It is easier to generalize rules to other similar stands, than to do so for a lookup table. It is also easier to add new rules or new inputs (such as roll acceleration) in order to make the compensation more precise, once the knowledge has been extracted.

Current Solutions. The current solution uses experimental data in the form of a lookup table, or some low-order polynomial approximation to it. This is not very accurate, but a high-order polynomial generalizes poorly. A neural network could be used for the task but the dataset available is small; the net architecture involves ad hoc choices; and the mapping so obtained would be opaque. Therefore, extending it or applying it to other similar situations would be difficult. This extendibility is also a problem for polynomials.

8.2.2 Solution Description

We use a fuzzy logic inference system to compute the compensation. This contains a small set of rules which captures the nonlinearities in the mapping. Instead of an expert for the rulebase, some training data is available. Therefore, we need a methodology that combines the positive features of both fuzzy logic and neural nets. It should capture concise and interpretable knowledge automatically from data.

Architecture Type. The architecture we use is that of a fuzzy logic inference system implemented as a multilayer, feedforward neural network. One specific example of such a scheme is ANFIS (Jang, 1993). The ANFIS architecture consists of a 6-layer neural network. Each layer and node has a concrete interpretation in terms of fuzzy inference, which makes the choice of hidden layer widths easy to initialize, interpret, and modify. Figure 15(a) shows one configuration used for the oil film compensation problem.

Structure and Parameters. The six layers implement fuzzy inference for TSK-type rules. Antecedent memberships have a soft trapezoidal shape, whereas the consequents are crisp linear functions of the two inputs. The free parameters (membership functions and consequent coefficients) are modified iteratively using least mean squared (LMS) optimization and backpropagation.

We propose using two to three membership functions per input dimension for this problem, since we want to avoid overfitting on the small dataset. The ANFIS architecture can be initialized using the fuzzy partition of input space. Data is normalized to lie in the unit hypercube, so that learning rates and other meta-parameters need not be scaled. The architecture is simple and trains quickly. It provides a small number of rules which can be interpreted linguistically. The key advantage of this approach is in the use of a limited training set to form a smooth, interpolating function.

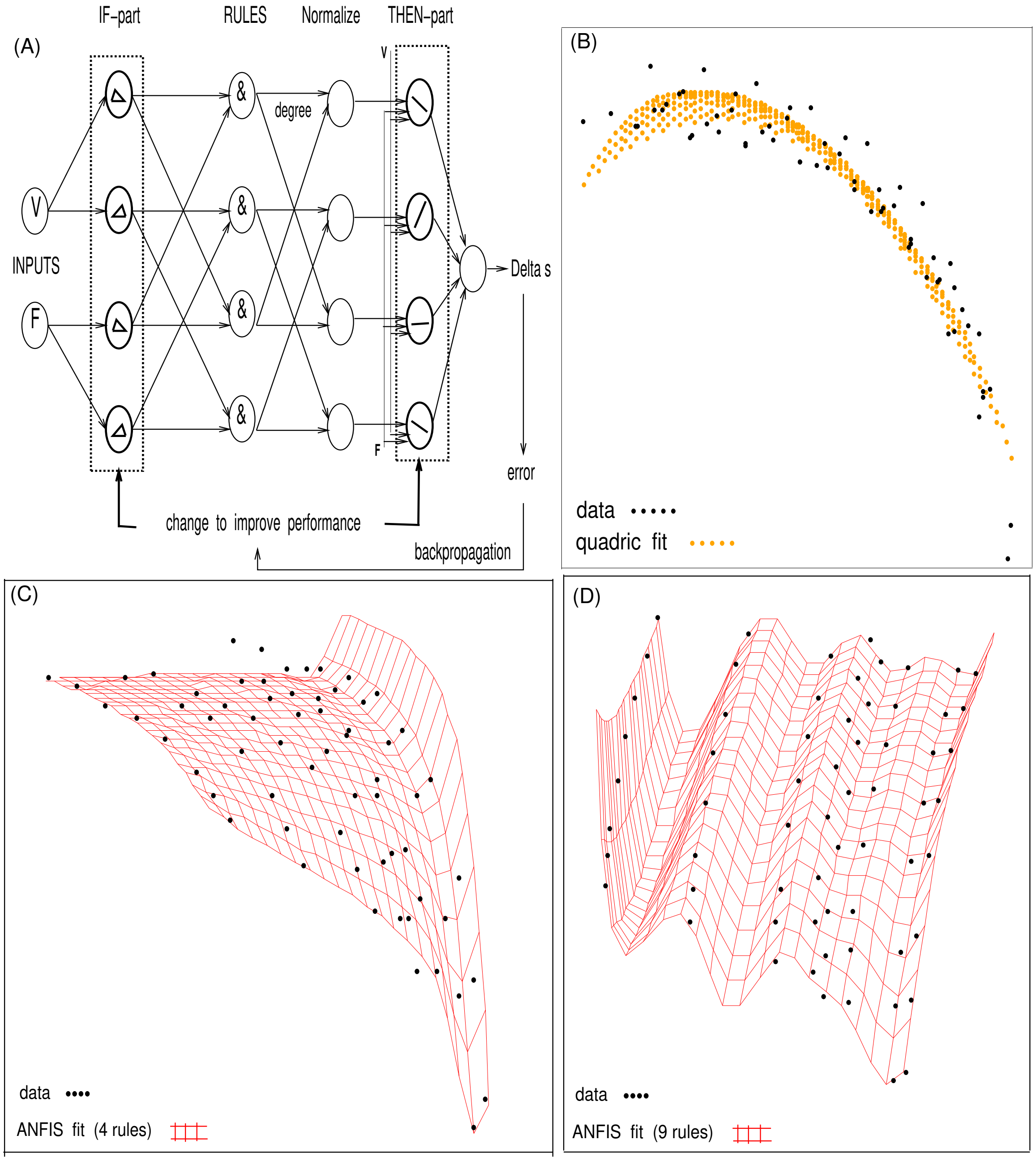

Figure 15: (a) The ANFIS architecture for the oil film compensation problem. (b) The quadric fit is not very accurate. (c) and (d) show ANFIS fits with 4 and 9 rules respectively. The latter is more complex at the expense of a better fit.

8.2.3 Results and Analysis

The two inputs (speed and force) determine Δs. Two different granularities for the neuro-fuzzy system are compared with polynomial fits. The fit metric is minimum least squares error. Most of the data is close to a smooth surface which is flat in one region and slopes steeply in another region, though there are significant deviations from this general rule.

Polynomial fits. The quadric is a bivariate, second-degree polynomial which fits the data with a minimum RMS error of 0.06278 per data point over the 64 points. Figure 15(b) shows that the fit is smooth but quite inaccurate as six degrees of freedom are not adequate to capture this particular nonlinearity.

ANFIS fits. The ANFIS system converges rapidly to a solution having an RMS error of 0.033816 for a 2x2 grid of 4 rules. Figure 15(c) shows that the surface can have varying curvature in different regions and still be reasonably smooth. However, one set of points forms a ridge in the data which accounts for most of the error of the ANFIS surface. A more complex model (a 3x3 grid of 9 rules) gives lower training of 0.019647, but the surface in Figure 15(d) has larger variation and worse generalization in order to get a better training accuracy.

Analysis. The deterioration with higher complexity is a manifestation of overfitting. In this problem, a 4-rule ANFIS strikes a reasonably good balance. If so desired, a 9-rule ANFIS may be slightly better, but it shows some oscillations in the surface. The linguistic rulebase engineered from the data is easier to understand and modify than a pure neural network or a polynomial model, due to the use of fuzzy sets and fuzzy inference.

We have also tried other approaches such as clustering using fuzzy spheres and ellipsoids to extract the rules. However, these may lead to a higher number of rules which are not always easy to interpret cognitively.

8.2.4 Application Conclusions

In summary, ANFIS extracted a few rules from data, providing transparency, smoothness, representation of prior knowledge, and learning capability. This neural-fuzzy combination gave high accuracy with a highly efficient training procedure. This procedure also leads itself to further fine-tuning, additions to the rulebase, and the number of inputs considered. This is a great improvement over the current polynomial-fit solution which is sensitive to small changes in the data.

The most important benefit of this reverse-engineered model is the potential reduction of the data set required to derive a model for other rolling mill stands. The first model derived can be used to initialize ANFIS for other stands, and the variations between stands can be accounted for with very little additional data. Measurements can be concentrated on steep regions of the surface while ignoring flat regions. The number of sampling points is thus reduced considerably. We are proposing to validate the results and applicability of this method and its derived model with new data sets. Furthermore we would like to apply this reverse-engineering technique to other "legacy-tables" which are currently used to represent unknown nonlinear relationships among process variables in tabular form.

9 High Throughput Requirements: Power Electronics Control

Power electronics requires a mental shift of scale: actions that previously had taken seconds or milliseconds to compute and implement are now required in microseconds; actuator values may vary by two or more orders of magnitude from one end of the range of operation to the other. Simulations using interpreted fuzzy logic controllers exhibited acceptable performance, but were not realizable in hardware because of throughput constraints. With the compiler technology mentioned above, we can approximate the controller's output with a look-up table and realize the controller using conventional hardware. Alternatively, special purpose hardware can be used to realize the fuzzy controller.

9.1 Resonant Converter Control for Power Supplies

9.1.1 Problem Description

Power supplies require a regulator to maintain the output voltage or output power constant in light of operational or environmental changes. For instance, the voltage of a power supply will tend to drop if load current increases. Similarly, changes in temperature affect the delivered power output.

Most resonant converters experience a high degree of nonlinearity in their control characteristics. Traditional control methods have used linear controllers while sacrificing potential electrical performance of the converters. Other approaches use complex nonlinear controls to obtain better performance; this often requires expensive current and voltage sensors to monitor electrical state variables within a given resonant converter.

A FLC provides an inexpensive nonlinear controller for obtaining good electrical performance. Such an FLC can exhibit increased robustness in the face of changing circuit parameters, saturation effects, or external disturbances.

The series resonant converter (SRC) in Figure 16 is a typical example of a nonlinear resonant system.

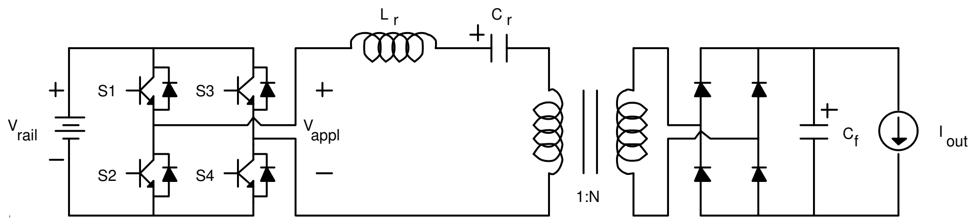

Figure 16: Circuit diagram of series resonant converter.

The class of resonant converters consists of converters using inductor (L) and capacitor (C) elements to assist in the switching operation of semiconductor devices. The additional L and C devices allow the semiconductor devices to operate in a zero-voltage switching (ZVS) or zero-current switching (ZCS) mode of operation. Many resonant converters have the additional advantage of lower electromagnetic interference (EMI) than traditional pulse-width modulated (PWM) converters. While resonant converters have the above advantages, they suffer from the disadvantage of difficulty in control.

A common resonant converter in usage today is the series resonant converter. Figure 16 shows a schematic of such a converter; it is *series-resonant* because the resonant tank elements (L_r and C_r) are in series with the electrical load. The semiconductor switches for this particular circuit can operate in one of two modes: always below the resonant tank natural frequency (subresonant), or always above the natural frequency (superresonant). For our purposes, we operate the series resonant converter in the superresonant mode of operation.

As stated earlier, the series resonant converter has many advantages over PWM converters, but suffers from the difficulty of a more complicated control method. While many PWM converters have very simple linear transfer functions, the non-linear transfer function of the series resonant converter requires a corresponding non-linear control algorithm to utilize the full capabilities of the converter.

The state equations for the series resonant converter are:

$$\dot{I_{L_r}} = \frac{1}{L_r}[V_{appl} - V_{C_r} - sign(I_{L_r})\frac{V_{C_f}}{N}]$$

$$\dot{V_{C_r}} = \frac{1}{C_r} I_{L_r}$$

$$\dot{V_{C_f}} = \frac{1}{C_f}[\frac{abs(I_{L_r})}{N} - I_{out}]$$

The absolute value and sign functions create a highly nonlinear transfer function. This causes severe control problems.

The approximate transfer function of the converter using a first harmonic analysis (Steigerwald, 1987) is

$$\frac{V_{C_f}}{V_{rail}} = \frac{1}{1 + j\frac{\pi^2}{8}\sqrt{\frac{L_r}{C_r}}\frac{V_{C_f}}{I_{out}}[\omega\sqrt{L_r C_r} - \frac{1}{\omega\sqrt{L_r C_r}}]}$$

Current Solutions. Many methods have been proposed to control the output voltage of the series resonant converter. These range from a simple proportional-integral controller closed around the output voltage, to elegant control methods (Oruganti and Lee, 1984, Sanders *et al.*, 1989, Schwarz, 1975).

The lack of a low cost, good transient response control circuit is a major obstacle in the widespread use of resonant converters. Our fuzzy logic control solution has the demonstrated advantage of very good control characteristics, while still maintaining low cost and allowing for use of inexpensive sensors.

An example of a control method that provides good system response is that of (Oruganti and Lee, 1984). This method provides good transient and steady state system response, but requires many expensive high frequency sensors. It is critical to properly tune the complex multiloop system. It is not very robust with respect to device characteristics that can change with operating conditions (e.g. resonant inductor saturation). The method is also very complex, and requires state plane methods to describe the converter control.

Methods of controlling the converter using in place circuit averaging (i.e. fourier) methods have been used for control(Sanders *et al.*, 1989). These methods are computationally expensive, and provide less accuracy than optimal control. In place averaging becomes less accurate as the converter switching function increases.

Some proposed control methods suffer from undesirable oscillatory behavior under certain operating conditions. An example is ADSTIC control(Schwarz, 1975). This method monitors the average current in the resonant inductor against a reference signal. The control problem arise when the instantaneous inductor current changes much faster than the control system can compensate.

There are many other control methods for a series resonant converter including capacitor voltage control(Ranganathan *et al.*, 1982), diode conduction control(King and Stuart, 1983), and many others.

9.1.2 Solution Description

Control Strategy. To determine the basic control law for this circuit, consider the case where the control frequency, or F_c, is restricted to values above the circuit's resonant frequency. As F_c is lowered towards the resonant frequency, the amount of energy available to the output capacitor increases, leading to a higher output voltage.

If the circuit were to be controlled in an open loop fashion, only a mapping from output voltage to F_c would be needed. However, the step response of the circuit degrades as F_c approaches the resonant frequency, overshoot and output ripple increasing markedly. This demonstrates the need for some sort of closed loop control.

The use of a linear controller leads to poor electrical performance at load and input voltage variations. Because the linear controller has been optimized for a particular operating condition, it exhibits wide variations in overshoot, rise time, and output ripple for different operating points.

The changing gain of the resonant converter is compensated by the nonlinear structure of the FLC. This provides improved converter operation largely independent of line and load variations.

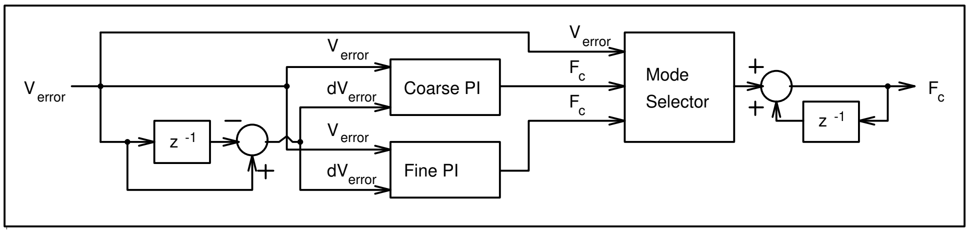

Figure 17: Hierarchical control scheme for series resonant converter.

9.1.3 Proposed FLC

Our approach uses this development environment and can be described in three stages: KB generation, KB compilation, and microcontroller implementation.

KB Generation. To close the loop, a hierarchical scheme, composed of two low level fuzzy logic controllers and a high level mode selector, is used to control the series resonant converter, as can be seen in Figure 17.

	NVL	NL	NM	NS	ZE	PS	PM	PL	PVL
PVL	PVL	NS	NM	NM	NL	NVL	NVL	NVL	NVL
PL	PVL	ZE	NS	NM	NM	NL	NL	NVL	NVL
PM	PVL	PS	ZE	NS	NM	NM	NM	NL	NVL
PS	PVL	PM	PS	ZE	NS	NM	NM	NL	NVL
ZE	PVL	PM	PM	PS	ZE	NS	NM	NM	NVL
NS	PVL	PL	PM	PM	PS	ZE	NS	NM	NVL
NM	PVL	PL	PM	PM	PM	PS	ZE	NS	NVL
NL	PVL	PVL	PL	PL	PM	PM	PS	ZE	NVL
NVL	PVL	PVL	PVL	PVL	PL	PM	PM	PS	NVL

Table 1: Ruleset for coarse controller. V_{error} along horizontal axis and dV_{error} along vertical axis.

	NVL	NL	NM	NS	ZE	PS	PM	PL	PVL
PVL	ZE	NS	NM	NM	NL	NVL	NVL	NVL	NVL
PL	PS	ZE	NS	NM	NM	NL	NL	NVL	NVL
PM	PM	PS	ZE	NS	NM	NM	NM	NL	NVL
PS	PM	PM	PS	ZE	NS	NM	NM	NL	NVL
ZE	PL	PM	PM	PS	ZE	NS	NM	NM	NL
NS	PVL	PL	PM	PM	PS	ZE	NS	NM	NM
NM	PVL	PL	PM	PM	PM	PS	ZE	NS	NM
NL	PVL	PVL	PL	PL	PM	PM	PS	ZE	PS
NVL	PVL	PVL	PVL	PVL	PL	PM	PM	PS	ZE

Table 2: Ruleset for fine controller. V_{error} along horizontal axis and dV_{error} along vertical axis.

Both low level controllers are fuzzy logic proportional integral (PI) controllers; by appropriately adjusting the scaling factors, membership functions, and rule sets of the fuzzy PIs, one has been configured as a coarse controller, and the other as a fine controller.

The inputs of both low level fuzzy logic PI controllers are: output voltage error (V_{error}) and the change in output voltage error (dV_{error}), where V_{error} is the difference of output voltage and output voltage setpoint and dV_{error} is the difference of the current and previous values of V_{error}. Both controllers output a change in the control frequency (dF_c). In accordance with the control strategy presented above, when the output voltage is below the setpoint and is continuing to decrease, the control frequency should decrease. In terms of V_{error}, dV_{error}, and dF_c, a large positive value of V_{error} and a large positive value of dV_{error} should result in a large negative value of dF_c. The coarse controller is specifically designed to reduce V_{error} at a rapid rate, while the fine controller is designed to maintain steady state error within acceptable values. Each has its own scaling factors for inputs and output.

This control law is encapsulated in the rule sets of Tables 1 and 2, where the three control variables have been partitioned into nine terms each: negative very large (NVL), negative large (NL), negative medium (NM), negative small (NS), zero (ZE), positive small (PS), positive medium (PM), positive large (PL), and positive very large (PVL). The rule set of the coarse controller (see Table 1) has two regions of large control action regardless of dV_{error}, corresponding to the columns under labels NVL and PVL. On the other hand, the rule set of the fine controller is similar to that of the MacVicar-Whelan rule set, as seen in Table 2.

To smoothly transition from coarse to fine control, a fuzzy logic mode selector is used to meld the output of the coarse and fine controls, based upon the magnitude of V_{error}. The coarse controller is dominant to within some percentage of the setpoint, at which point the mode selector begins to lower the weighting of the coarse controller, while increasing the weighting of the fine controller. When the output voltage is within a small percentage of the setpoint, the coarse controller is fully disabled, and the fine controller is allowed to reduce V_{error} to zero.

Once the mode selector has blended the outputs of the low level controllers, the resulting value of dF_c is then integrated to give a new control frequency. However, control frequencies must stay above the natural frequency of the resonant converter so that the basic control law remains valid, while remaining below a maximum practical switching frequency. Thus, F_c is clamped between a minimum and maximum value before being passed to the actuator.

KB Compilation. Once the KBs for the mode selector and both of the low level controllers have been determined, they are then compiled into fast look-up tables. To accomplish this, the hierarchical control scheme can be thought of as a single controller; iterating through the input space defined by V_{error} and dV_{error} and recording the resulting values of dF_c then gives a desired look-up table relating control inputs to control outputs.

Microcontroller Implementation. For an FLC-based frequency controller, if a control action is taken once in the period defined by $1/F_c$, and F_c operates in the region of hundreds of kiloHertz, control frequency actions must be determined within microseconds. In other resonant circuit topologies, the switching frequency is further increased to the megaHertz region, creating even tighter throughput requirements. Therefore, this implementation necessitates a relatively fast microcontroller.

Each time step, V_{error} and dV_{error} are calculated from sensor readings. The resulting values are used to locate the corresponding value of dF_c in the look-up table compiled above. After integrating dF_c and clamping the resulting control frequency, the controller then passes that value to the actuator.

9.1.4 Results, Analysis, and Conclusions

Performance Traditional power convert controls use fixed gain compensators. Typically they exhibit slow system dynamic responses as a price for ensuring stability under all load conditions.

On the other hand, FLC controllers have much faster dynamic responses, while maintaining stable performance under all load conditions. These results are due to the FLC's flexible design architecture: the power converters are optimized for different operating points (with different corresponding low-level nonlinear controllers); for a given state, the hierarchical controller decides their level of applicability and provides a smooth interpolation of the outputs of the low-level controllers.

SRC Results The fuzzy logic controller exhibits good performance for a number of different operating conditions. We used a sequence of eight setpoints starting at 125KV and ramping down to 50KV (typical X-ray machine voltage regimes). In all cases, we observed very short rise times followed by minimal or no overshooting/undershooting.

Implementation The development of a FLC using a development environment and a compiler (Bonissone, 1991b) results in a general architecture composed of an address generator, a lookup table ROM, and a fast microprocessor with limited amount of RAM. Following this concept we implemented a hardware prototype of the fuzzy controller for a more complex circuit topology, the Single-Ended Parallel Multi-Resonant Converter (SEP-MRC), with a switching frequency of about 500KHz. This demonstrates the generality of the FLC architecture, in which the control of other resonant converter topologies can be achieved by simply replacing the content of the compiled lookup table. A more detailed description of the SEP-MRC application can be found in reference (Bonissone *et al.*, 1985).

10 PART II Conclusions: FLC Applications

10.1 Summary

From a sample of our own experience, partially described in Sections 7, 8, and 9, we have described the synthesis of nonlinear controllers for a variety of dynamic systems, spanning a large space of cost, performance and throughput requirements.

We have emphasized the role of FLC at the supervisory level to provide smooth mode melding (instead of mode selection) by choosing among low-level conventional controllers (Badami *et al.*, 1994, Bonissone and Chiang, 1995). or to provide a nonlinear gain schedule for conventional low-level PID controllers(Zhe *et al.*, 1993). We have shown the synergy resulting from combining FLC with conventional control techniques, such as optimal control or with data-driven learning techniques, such as back-propagation. Finally, we have described the use of FLCs in high throughput applications, such as power electronics, and we have illustrated the role of a FLC compiler to realize the controller using off-the-shelf hardware.

10.2 Comments

We have demonstrated that FLC technology is a powerful control synthesis technique that complements other analytical or data-driven techniques. We have discussed FLCs' applicability to a vast gamut of applications, spanning a large range of cost, performance, and throughput requirements.

We will now focus on the combination of FLCs with other emerging technologies, such as neural network and genetic algorithms, that will further improve the FLCs cost/benefit ratio when applied to many complex control problems.

11 PART III: Soft Computing and Approximate Reasoning Systems

Soft Computing (SC) is a recently coined term describing the symbiotic use of many emerging computing disciplines. According to Lotfi Zadeh (Zadeh, 1994): *"... in contrast to traditional, hard computing, soft computing is tolerant of imprecision, uncertainty, and partial truth."* In the context of our discussion we will consider Fuzzy Logic (FL), Probabilistic Reasoning (PR), Neural Networks (NNs), and Genetic Algorithms (GAs) as Soft Computing main components.

Fuzzy Logic (Zadeh, 1965) gives us a language, with syntax and local semantics, in which we can translate our qualitative knowledge about the problem to be solved. FL's main characteristic is the robustness of its interpolative reasoning mechanism.

Probabilistic Reasoning, such as Bayesian Belief Networks (Bayes, 1763) and Dempster-Shafer theory of belief (Dempster, 1967, Shafer, 1976), gives us the mechanism to evaluate the outcome of systems affected by randomness or other types of probabilistic uncertainty. PR's main characteristic is its ability to update previous outcome estimates by conditioning them with newly available evidence.

Neural Networks are computational structure that can be trained to learn patterns from examples (Widrow and Hoff, 1960). By using a training set, which samples the relation between inputs and outputs, and a back-propagation type of algorithm (Werbos, 1974), NNs give us a supervised learning algorithm that performs fine-granule local optimization.

Genetic Algorithms (GAs) (Holland, 1975) give us a way to perform randomized global search in a solution space. In this space, a population of candidate solutions, encoded as chromosomes, is evaluated by a fitness function in terms of its performance. The best candidates *evolve* and pass some of their characteristics to their *offsprings.*

The common denominator of these technologies is their departure from classical reasoning and modeling approaches that are usually based on boolean logic, analytical models, crisp classifications, and deterministic search. In ideal problem formulations, the systems to be modeled or controlled are described by complete and precise information. In these cases, formal reasoning systems (theorem provers) can be used to attach binary truth values to statements describing the state or behavior of the physical system. This is illustrated in Figure 18.

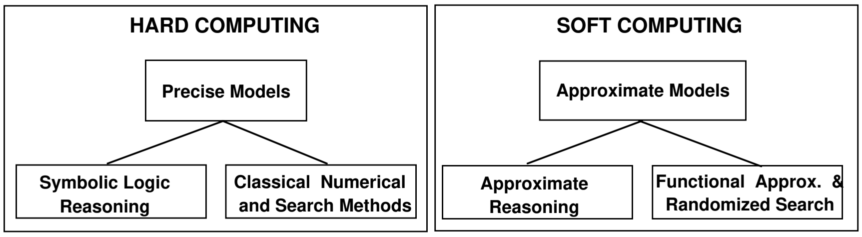

Figure 18: Hard and Soft Computing

As we attempt to solve real-world problems, however, we realize that they are typically ill-defined systems, difficult to model and with large-scale solution spaces. In these cases, precise models are impractical, too expensive, or non-existent. The relevant available information is usually in the form of empirical prior knowledge and input-output data representing instances of the system's behavior. Therefore, we need approximate reasoning systems capable of handling such imperfect information. Soft Computing technologies provide us with a set of flexible computing tools to perform these approximate reasoning and search tasks.

In the remaining of this report we will describe and contrast Soft Computing technology components: fuzzy and probabilistic reasoning, neural networks, and genetic algorithms. Then we will illustrate some examples of hybrid systems developed by leveraging combinations of these components, such as the control of GAs and NNs parameters by FL; the evolution of NNs topologies and weights by GAs or its application to tune FL controllers; and the realization of FL controllers as NNs tuned by backpropagation-type algorithms. Figure 19 provides a graphical summary of the these hybrid algorithms and their components. The interest reader should consult reference (Bouchon-Meunier *et al.*, 1995) for an extensive coverage of this topic.

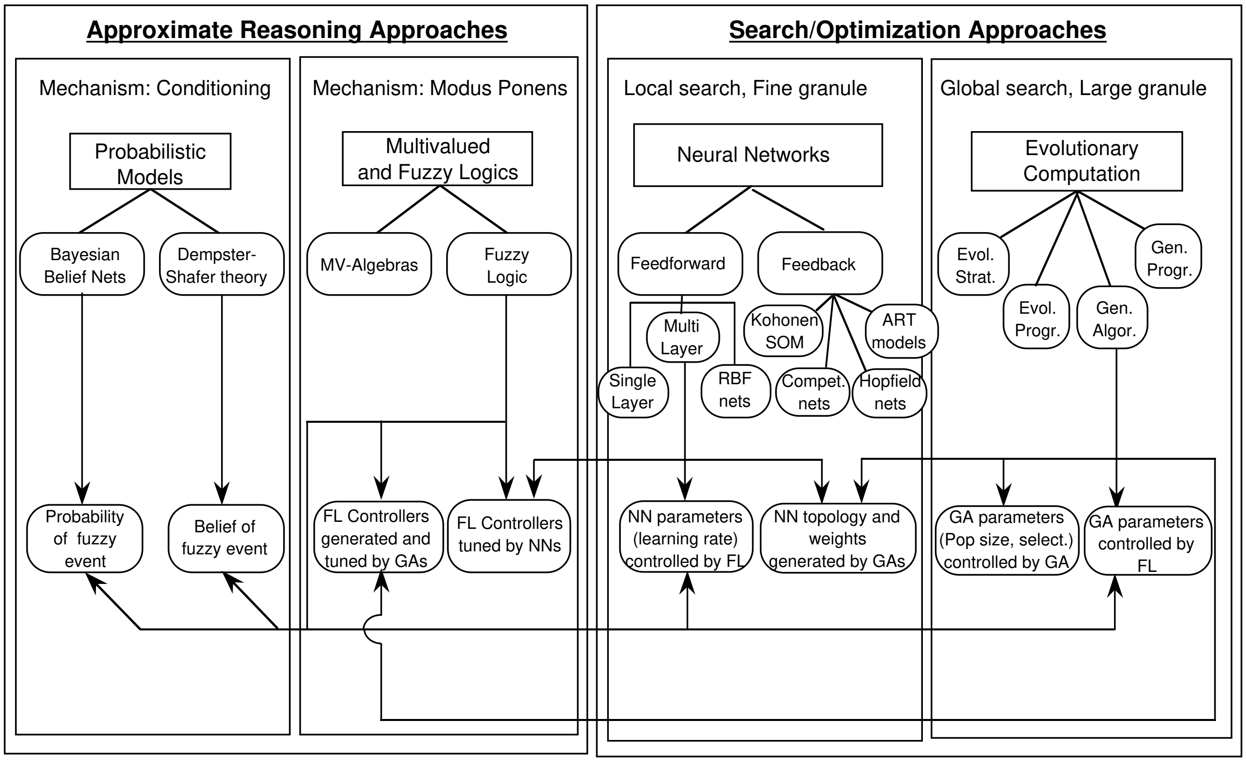

Figure 19: Soft Computing Overview

12 Probability and Fuzziness

12.1 Distinctions

Randomness and fuzziness capture two rather different types of uncertainty and imprecision. In *randomness*, the uncertainty is derived by the nondeterministic membership of a point from the sample space in a well-defined region of that space. The sample space describes the set of possible values for the random variable. The point is the outcome of the system. The well-defined region represents the event whose probability we want to predict. The characteristic function of the region dichotomizes the sample space: either the point falls within the boundary of the region, in which case its membership value in the region is one and the event is true, or it falls outside the region, in which case its membership value in the region is zero and the event is false. A probability value describes the tendency or frequency with which the random variable takes values inside the region.

On the other hand, in *fuzziness* the uncertainty is derived from the partial membership of a point from the universe of discourse in an imprecisely defined region of that space. The region represents a fuzzy set. The characteristic function of the fuzzy set does not create a dichotomy in the universe of discourse. It defines a mapping from such universe into the real-valued interval [0,1] instead of the set {0,1}. A partial membership value does not represent any frequency. Rather, it describes the degree to which that particular element of the universe of discourse satisfies the property that characterizes the fuzzy set (Zadeh, 1965).

12.2 Interpretations

Not all probabilities have frequentistic interpretations. For example, subjective probabilities (DeFinetti, 1937) can be defined in terms of the willingness of a rational agent to accept a bet, in which the ratio of its associated cost and prize reflects the probability of the event. Similarly, fuzzy membership values may have more than one interpretation, ranging from *possibility* values (fuzzy restrictions that act as an elastic constraints on the value that may be assigned to a variable (Zadeh, 1978)), to *similarity* values (the complement of the distances among possible worlds) (Ruspini, 1989, Ruspini, 1990), to *desirability or preference* values (the partial order induced by the membership function on the universe of discourse) (Dubois and Prade, 1992).

12.3 Probabilistic Reasoning Systems

The earliest probabilistic techniques are based on single-valued representations. These techniques started from approximate methods, such as the modified Bayesian rule (Duda *et al.*, 1976) and confirmation theory (Shortliffe and Buchanan, 1975), and evolved into formal methods for propagating probability values over Bayesian Belief Networks (Pearl, 1982, Pearl, 1988b). Another trend among the probabilistic approaches is represented by interval-valued representations such as Dempster-Shafer theory (Dempster, 1967, Shafer, 1976, Smets, 1991). In all approaches, the basic inferential mechanism is the *conditioning* or updating operation. We will briefly review two main currents within probabilistic reasoning: Bayesian Belief Networks, and Dempster-Shafer's theory of belief.

12.3.1 Bayesian Belief Networks

Over the last ten years, considerable efforts have been devoted to improve the computational efficiency of Bayesian belief networks for trees, poly-trees, and Directed Acyclic Graphs (DAGs) such as influence diagrams (Howard and Matheson, 1984), (Schachter, 1986), (Agogino and Rege, 1987).

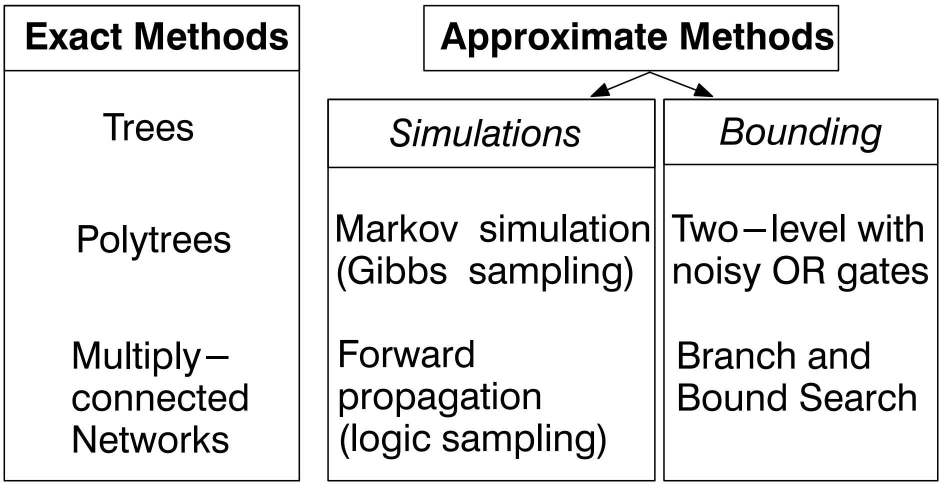

Figure 20: Taxonomy of Inference Mechanisms for Bayesian Belief Networks

An efficient propagation of belief on Bayesian Networks has been originally proposed by J. Pearl (Pearl, 1982, Pearl, 1986a). I n his work, Pearl describes an efficient updating scheme for trees and, to a lesser extent, for poly-trees (Kim and Pearl, 1983, Pearl, 1986b, Pearl, 1988a). However, as the graph complexity increases from trees to poly-trees to general graphs, so does the computational complexity. The complexity for trees is $O(n^2)$, where n is the number of values per node in the tree. The complexity for poly-trees is $O(K^m)$, where K is the number of values per parent node and m is the number of parents per child. This number is the size of the table attached to each node. Since the table must be constructed manually and updated automatically, it is reasonable to assume that the the value of m will be small and so will the table. The complexity for multiconnected graphs is $O(K^n)$, where K is the number of values per node and n is the size of the largest nondecomposable subgraph. To handle such complexity, techniques such as moralization and propagation in a tree of cliques (Lauritzen and Spiegelhalter, 1988) and loop cutset conditioning (Suermondt *et al.*, 1991, Stillman, 1991) are typically used to decrease the value of n, decomposing the original problem represented by the graph into a set of smaller problems or subgraphs.

When this problem decomposition process is not possible, exact methods are abandoned in favor of approximate methods. Among these methods the most common are clustering, bounding conditioning (Horvitz *et al.*, 1989), and simulation techniques, such as logic samplings and Markov simulations (Henrion, 1989). This is illustrated in Figure 20

12.3.2 Dempster-Shafer Theory of Belief

Belief functions have been introduced in an axiomatic manner by Shafer (Shafer, 1976). Their original purpose was to compute the degree of belief of statements made by different sources or witnesses from a subjective probability of the sources reliability.

Many other interpretations of belief functions have been presented, ranging from functions induced from a probability measure by multivalued mappings (Dempster, 1967) or by compatibility relations (Lowrance *et al.*, 1986), to probability of provability (Pearl, 1988a), to inner measures (Ruspini, 1987, Fagin and Halpern, 1989), to a nonprobabilistic model of transferable belief (Smets, 1991).

All interpretations share the same static component of the theory: the Möbius Transform, which defines a mapping from basic probability assignments, masses assigned to subsets of the frame of discernment, to the computation of the lower bound (belief) of a proposition, a region defined in the same frame of discernment. An inverse Möbius transform can be used to recover the masses from the belief. All these interpretations also share the same definition of the upper bound, usually referred to as plausibility.

More specifically, this formalism defines a function that maps subsets of a space of propositions Θ on the [0,1] scale. The sets of partial beliefs are represented by mass distributions of a unit of belief across the propositions in Θ. This distribution is called basic probability assignment (bpa). The total certainty over the space is 1. A non-zero *bpa* can be given to the entire space Θ to represent the degree of ignorance, which models the source lack of complete reliability. Given a space of propositions Θ, referred to as frame of discernment, a function $m : 2^\Theta \rightarrow [0,1]$ is called a basic probability assignment if it satisfies the following three conditions:

$$m(\phi) = 0 \quad \text{where } \phi \text{ is the empty set} \tag{12}$$

$$0 < m(A) < 1 \tag{13}$$

$$\sum_{A \subseteq \Theta} m(A) = 1 \tag{14}$$

The certainty of any proposition A is then represented by the interval $[Bel(A), P^*(A)]$, where $Bel(A)$ and $P^*(A)$ are defined as:

$$Bel(A) = \sum_{\phi \neq x \subseteq A} m(x) \tag{15}$$

$$P^*(A) = \sum_{x \cap A \neq \phi} m(x) \tag{16}$$

where $x \subseteq \Theta$. From the above definitions the following relation can be derived:

$$Bel(A) = 1 - P^*(\neg A) \tag{17}$$

Equations 15 and 16 represent the *static* component of the theory, which is common to all interpretations. However, these interpretations do not share the same *dynamic* component of the theory: the process of updating (i.e., conditioning or evidence combination). This issue has been recently addressed by various researchers (Halpern and Fagin, 1990, Smets, 1991).

As for the case of belief networks, a variety of exact and approximate methods have been proposed to perform inferences using belief functions. Typically, the exact methods require additional constraints on the structure of the evidence. Figure 21 illustrates a taxonomy of Dempster-Shafer inference mechanisms.

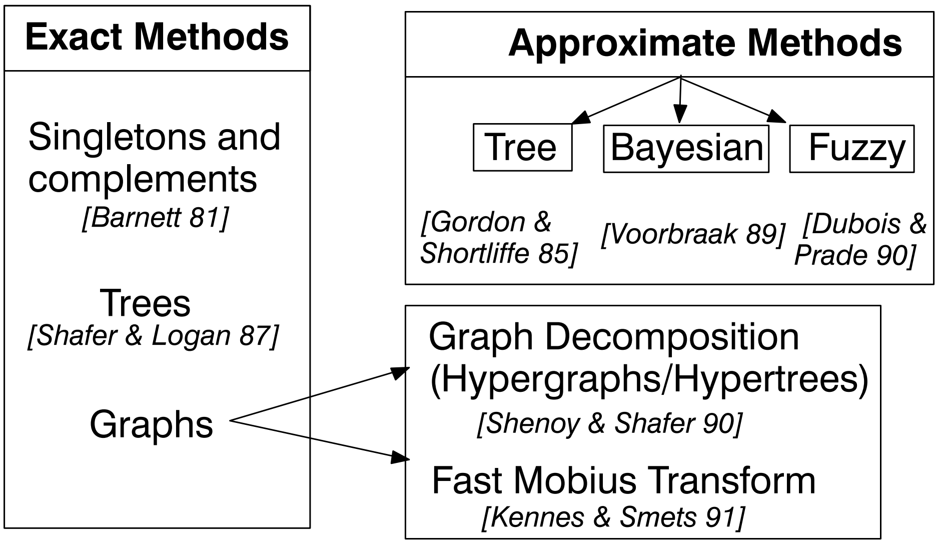

Figure 21: Taxonomy of Inference Mechanisms for Dempster-Shafer

Inference Mechanism: Conditioning Given the beliefs (or masses) for two propositions A and B, Dempster's rule of combination can be used, under assumptions of independence, to derive their combined belief (or mass).

If m_1 and m_2 are two *bpa*s induced from two independent sources, a third *bpa*, $m(C)$, expressing the pooling of the evidence from the two sources, can be computed by using Dempster's rule of combination:

$$m(C) = \frac{\sum\limits_{A_i \cap B_j = C} m_1(A_i) \cdot m_2(B_j)}{1 - \sum\limits_{A_i \cap B_j = \phi} m_1(A_i) \cdot m_2(B_j)} \tag{18}$$

Dempster's rule allows us to consider and pool discounted pieces of evidence, i.e. evidence whose belief can be less than one. On the other hand, conditioning can only be done with certain evidence. If proposition B is true (i.e., event B has occurred), then $Bel(B) = 1$ and from Demspster rule of combination, we can derive a formula for conditioning A given B, $Bel(A \mid B)$:

$$Bel(A \mid B) = \frac{Bel(A \cup \neg B) - Bel(\neg B)}{1 - Bel(\neg B)} \tag{19}$$

This expression is compatible with the interpretation of Belief as evidence, and as inner measure. However, this expression is not compatible with the interpretation of belief as the lower envelope of a family of probability distributions. Under such interpretation, the correct expression for conditioning is

$$Bel(A \parallel B) = \frac{Bel(A \cap B)}{Bel(A \cap B) + Pl(\neg A \cap B)} \tag{20}$$

The interested reader is referred to reference (Shafer, 1990) for a clear explanation and an updated bibliography on belief functions.

All above probabilistic methods use the operation of *conditioning* to update the probability values and perform a probabilistic inference. We will now switch our focus to fuzzy logic based systems, a class of approximate reasoning systems whose inference mechanism is not conditioning, but an extension of *modus-ponens*.

12.4 Fuzzy Logic Based Reasoning Systems

Fuzzy logic approaches are based on a fuzzy-valued representation of uncertainty and imprecision. Typically they use Linguistic Variables (Zadeh, 1978, Zadeh, 1979) to represent different information granularities and Triangular-norms to propagate the fuzzy boundaries of such granules (Schweizer and Sklar, 1963, Schweizer and Sklar, 1983, Dubois and Prade, 1984, Bonissone, 1987, Bonissone and Decker, 1986, Bonissone *et al.*, 1987b).

The basic inferential mechanism used in fuzzy reasoning systems is the *generalized modus-ponens* (Zadeh, 1979), which makes use of inferential chains (syllogisms).

12.4.1 Triangular norms: A Review

Since Triangular norms play such an important role in the definition of the generalized modus ponens, we will provide the reader with a brief overview of these operators. Triangular norms (T-norms) and their dual T-conorms are two-place functions from [0,1]x[0,1] to [0,1] that are monotonic, commutative and associative. They are the most general families of binary functions that satisfy the requirements of the conjunction and disjunction operators, respectively. Their corresponding boundary conditions satisfy the truth tables of the Boolean AND and OR operators.

Any triangular norm $T(A, B)$ falls in the interval $T_w(A, B) \leq T(A, B) \leq Min(A, B)$, where

$$T_w(A, B) = \begin{cases} \min(A, B) & \text{if } \max(A, B) = 1, \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

The corresponding DeMorgan dual T-conorm, denoted by $S(A, B)$, is defined as

$$S(A, B) = 1 - T(1 - A, 1 - B) \tag{22}$$

$T_w(A, B)$ is referred to as the *drastic* T-norm (to reflect its extreme behavior) and is clearly non-continuous. By changing one of the axioms of the T-norms (Schweizer and Sklar, 1963), we can derive a subset of T-norms, referred to as *copulas*, such that any copula $T(A, B)$ falls in the interval $Max(0, A + B - 1) \leq T(A, B) \leq Min(A, B)$.

In the original version of fuzzy logic proposed by Zadeh (Zadeh, 1965), the conjunction and disjunction operators are the *minimum* and *maximum*, i.e. the upper and lower bounds of the T-norm and T-conorm ranges, respectively. These operators are the only ones satisfying distributivity and idempotency (Bellman and Giertz, 1973). Other selection of T-norms and T-conorms provide different logics with different properties (Klement, 1981, Bonissone and Decker, 1986).

Perhaps the most notable selection is the one based on the lower bound of the T-norms (Lukasiewicz T-norm) (Lukasiewicz, 1967) and its dual T-conorm. This logic satisfies the *law of the excluded-middle* (at the expense of distributivity) and is the basis of MV-Algebras (Di Nola and Gerla, 1986). In the fuzzy logic community this algebra was originally referred to as *bold* algebra (Giles, 1981). Mundici (Mundici, 1995) has provided interesting semantics for this algebra, presenting it as a decision making (voting) paradigm in a context where the source of information is allowed to have up to a known maximum number of k lies (errors). This is also known as Ulam's game with k lies and has direct applications to on-line error correcting code.

Fuzzy reasoning systems can be used in many applications, from advice providing expert systems, to soft constraint propagation, to decision making systems, etc. Within this paper we will limit our scope to Fuzzy Controllers (FCs), reasoning systems composed of a Knowledge Base (KB), an inference engine, and a defuzzification stage. The KB is comprised by a rule base, describing the relationship between state vector and output, and by the semantics of the linguistic terms used in the rule base. The semantics are established by scaling factors delimiting the regions of saturation and by termsets defining a fuzzy partition in the state and output spaces (Bonissone and Chiang, 1993).

The concept of a fuzzy controller was initially outlined by Zadeh (Zadeh, 1973) and first explored by Mamdani (Mamdani and Assilian, 1975, Kickert and Mamdani, 1978) in the early seventies. Currently it represents one of the most successful applications of fuzzy logic based systems (Bonissone *et al.*, 1995).

12.5 Complementarity

The distinction between probability and fuzziness has been presented and analyzed in many different publications, such as (Bezdek, 1994, Dubois and Prade, 1993, Klir and Folger, 1988) to mention a few. Most researches in probabilistic reasoning and fuzzy logic have reached the same conclusion about the complementarity of the two theories (Bonissone, 1991a). This complementarity was first noted by Zadeh (Zadeh, 1968), who, in 1968, introduced the concept of the probability measure of a fuzzy event. Let A be a fuzzy event, i.e, a subset of a finite sample space X. Let also x_i represent the ith singleton in such a space and $p(x_i)$ be its probability. Then $P(A)$, the probability of the fuzzy event A, is the weighted sum of the probability of each singleton in the sample space multiplied by $\mu_A(x_i)$, the partial degree to which the singleton x_i belongs to the fuzzy subset A, i.e.:

$$P(A) = \sum_{x_i \in X} p(x_i) \times \mu_A(x_i) \tag{23}$$

In 1981 Smets, extended the theory of belief functions to fuzzy sets by defining the belief of a fuzzy event (Smets, 1981, Smets, 1988). Let A be a fuzzy event of a finite sample space X. Let also S_i represent the ith available piece of evidence, i.e. a non-empty subset of the frame of discernment with an assigned probability mass $m(S_i)$. Then $Bel(A)$, the belief of the fuzzy event A, is the weighted sum of the probability masses of each piece of evidence multiplied by the minimum degree to which the evidence supports the event, i.e. it is included in the fuzzy region defining the event:

$$Bel(A) = \sum_{\phi \neq S_i \subseteq X} m(S_i) \times \bigwedge_{x_j \in S_i} \mu_A(x_j) \tag{24}$$

We have established the orthogonality and complementarity between probabilistic and possibilistic methods. Given their duality of purpose and characteristics, it is clear that these technologies ought to be regarded as being complementary rather than competitive.

13 Neural Networks

Fuzzy logic enables us to translate and embed empirical, qualitative knowledge about the problem to be solved into reasoning systems capable of performing approximate pattern matching and interpolation. Fuzzy logic however does not have adaptation or learning features, since it lacks the mechanism to extract knowledge from existing data. Of course, it could be argued that it is possible to use fuzzy clustering methods, such as Fuzzy C-means (Bezdek and Harris, 1978, Bezdek, 1981) to provide more accurate definitions of the membership functions of the state and output variables, in a typical unsupervised mode. However, FL systems are not able to *learn* from examples of input-output pairs, in a typical supervised mode.

On the other hand, this is the typical characteristic of Neural Networks, another Soft Computing technology. NNs and Perceptorns started in the early 60s as algorithms to train adaptive elements. Their origins can be traced to the work of Rosenblatt on *spontaneous learning* (Rosenbaltt, 1959), Stark's work on competitive learning (Stark *et al.*, 1962) and Widrow's development of ADALINE (Widrow and Hoff, 1960) and MADALINE algorithms.

Typically NNs are divided into *Feed-Forward* and *Recurrent/Feedback* networks. The Feed-Forward networks include single-layer perceptrons, multilayer perceptrons, and Radial Basis function nets (RBFs) (Moody and Darken, 1989), while the Recurrent nets cover Competitive networks, Kohonens Self Organizing Maps (Kohonen, 1982), Hopfield nets (Hopfield, 1982), and ART models (Carpenter and Grossberg, 1983, Carpenter and Grossberg, 1987, Carpenter and Grossberg, 1990). While feedforward NNs are used in supervised mode, recurrent NNs are typically geared toward unsupervised learning, associative memory, and self-organization. In the context of our paper we will only consider feedforward NNs. Given the functional equivalence already proven between RBF and fuzzy systems (Jang *et al.*, 1993) we will further limit our discussion to multilayer feedforward nets.

A feedforward multilayer NN is composed of a network of processing units or neurons. Each neuron performs the weighted sum of its input, using the resulting sum as the argument of a non-linear activation function. Originally the activation functions were sharp thresholds (or Heavyside) functions, which evolved to piecewise linear saturation functions, to differentiable saturation functions (or sigmoids), and to gaussian functions (for RBFs). For a given interconnection topology, NNs train their weight vector to minimize a quadratic error function.

Prior to backpropagation (Werbos, 1974) there was no sound theoretical way to train multilayers, feedforward networks with nonlinear neurons. On the other hand single-layer NNs (perceptrons) were too limited, as they could only provide linear partitions of the decision space. While their limitations were evidenced by Minsky and Papert (Minsky and Papert, 1969), Hornik et al. proved that a three-layers NN were universal functional approximators (Hornick *et al.*, 1989). Therefore, the advent of BP made multilayers feedforward NNs extremely popular. Since then, most of the research work on NNs has been devoted to improve their converge speed: by using estimates of the second derivatives, under simplifying assumptions of a quadratic error surface, as in Quickprop (Fahlman, 1988); by changing the size of the step size in a self-adapting fashion such as SuperSAB (Tollenaere, 1990); or by using second order information, as in in the Conjugate Gradient Descent method, (Moller, 1990). An excellent history of Adaptive NNs is provided by Widrow in reference (Widrow, 990).

13.1 Learning

In the context of this paper, we will consider learning only in the context of Soft Computing. Therefore, we will limit our discussion to *structural* and *parametric* learning, which are the counterpart of system identification and parameter estimation in classical system theory. For a fuzzy controller learning (or tuning) entails defining (or refining) the knowledge base (KB), which is composed of the a parameter set (state and output scaling factors, state and output termsets) and a structure (the rule base). The parameter set describe the local semantics of the language and the rule set describe the syntactic mapping. For Neural Networks, structural learning means the synthesis of the network topology (i.e., the number of hidden layers and nodes), while parametric learning implies determining the weight vectors that are associated to each link in a given topology.

Learning can be facilitated by the availability of complete or partial feedback. In the case of total feedback (a teacher providing an evaluation at every iteration or a training set describing the correct output for a given input vector) we have supervised learning. When only partial feedback is available (every so often we are told if we succeed or failed) we have reinforcement learning. When no feedback is available we have the case of unsupervised learning.

13.1.1 Supervised Learning

In the context of supervised learning, ANFIS (Adaptive Neural Fuzzy Inference Systems) (Jang, 1993) is a great example of an architecture for tuning fuzzy system parameters from input-output pairs of data. The fuzzy inference process is implemented as a generalized neural network, which is then tuned by gradient descent techniques. It is capable of tuning antecedent parameters as well as consequent parameters of TSK-rules which use a softened trapezoidal membership function. It has been applied to a variety of problems, including chaotic timeseries prediction and the IRIS cluster learning problem. As a fuzzy system, it does not require a large data set and it provides transparency, smoothness, and representation of prior knowledge. As a neural system, it provides parametric adaptability.

13.1.2 Steepest Descent

Backpropagation neural-net based techniques usually depend on a differentiable input-output map, which restricts their applicability. For controllers, it is practical to evaluate their performance over the whole trajectory rather than individual states, and a few such evaluations provide a crude local snapshot of the performance surface as a function over parameter space. This snapshot can then guide a steepest descent algorithm to determine the two scaling factors (K_p and K_i) in the design of a fuzzy logic PI controller.

The evaluation function is a metric based on the total deviation of the actual trajectory from an ideal trajectory, which is crafted based on the specifications of the controller, such as rise time, settling time, steady-state error band and steady-state oscillation. The metric can be quite flexible if desired.

The method uses a logarithmic search of the parameter space followed by a linear one to identify the appropriate scaling factors. The same method can be applied to other critical parameters such as the centers of the output membership functions. For low-dimensional searches, this method can be applied easily to any kind of system and remains reasonably efficient, since it is easy to parallelize.

13.1.3 Reinforcement Learning

Reinforcement learning exploits the availability of expert knowledge in the area of exerting control actions as well as evaluating system state. Approximate linguistic rules can be used to initialize the two knowledge bases which deal with action selection and action evaluation. The resulting system is capable of learning to control a complex, dynamic system in the absence of desired output, with only a delayed, somewhat uninformative reinforcement signal from the environment. This system has been used to control systems from the inverted pendulum to the space shuttle's attitude control (Berenji and Khedkar, 1992)

13.1.4 Structural Learning: Rule Clustering

The previously mentioned systems deal mainly with parameter identification once the structure has been fixed. However, identifying the number of rules in a fuzzy system or fixing the granularity of the fuzzy partition of the input space is a structure identification problem which also needs to be solved. If expert rules are not available, then other known properties of the unknown function may be available and could be exploited. For instance, in industrial settings, many mappings are implemented as approximations using look-up tables or analytic interpolation functions. If a fuzzy system can be reverse engineered from such information, then knowledge extraction can help to refine or upgrade the system.

If analytical information about the mapping is available, then various algorithms can be used to extract a near-optimal fuzzy rulebase which is equivalent to the mapping. On the other hand, the function can be sampled for data points or the look-up table can be used to generate the data points according to a desired distribution. If there is sufficient data, it is possible to adapt neural network clustering methods to extract clusters in product space which correspond to fuzzy rules. The method uses the joint criteria of incompleteness as well as accuracy in prediction to add rules to the database and then conducts a deletion phase to prune redundant knowledge. This system extracts a set of rules from a single online pass over a reasonably small dataset. It can then be tuned by using the same gradient optimization techniques to tune the parameters as have been discussed above. The reader is referred to (Berenji and Khedkar, 1993, Khedkar, 1993) for a detailed discussion of reinforcement learning and rule clustering.

14 Evolutionary Computing

In the previous section we discussed supervised learning of fuzzy system parameters. Since gradient descent techniques may become mired in local minima, global search techniques have also been explored. We will focus our attention on a randomized global search paradigm, which is commonly referred to as Evolutionary Computation (EC). This paradigm covers several variations, such as *Evolutionary Strategies* (ES), addressing continuous function optimization (Rechenberg, 1965, Schwefel, 1965); *Evolutionary Programs* (EP), generating finite state automata that describe strategies or behaviors (Fogel, 1962, Fogel *et al.*, 1966); *Genetic Algorithms* (GAs), providing continuous and discrete function optimization, system synthesis, tuning, testing, etc. (Holland, 1975); and *Genetic Programming* (GP), evolving computer programs to approximately solve problems, such as generating executable expressions to predict timeseries, etc. (Koza, 1992).

As noted by Fogel ((Fogel, 1995), page 103) in his historical perspective and a comparison of these paradigms: *...the three main lines of investigation - genetic algorithms, evolution strategies, and evolutionary programming - share many similarities. Each maintains a population of trial solutions, imposes random changes to those solutions, and incorporate selection to determine which solutions to maintain in future generations....* Fogel also notes that GAs *emphasize models of genetic operators as observed in nature, such as crossing-over, inversion, and point mutation, and apply these to abstracted chromosomes.* while ES and EP *emphasize mutational transformations that maintain behavioral linkage between each parent and its offspring.* In this paper, we will limit our analysis to Genetic Algorithms.

14.1 Genetic Algorithms

Genetic Algorithms (GAs) are perhaps the most widely known of the above paradigms. In the context of designing fuzzy controllers, it is relatively easy to specify an evaluation of the trajectory or the controller as a whole, but it is difficult to specify desired step-by-step actions, as would be required by supervised learning methods. Thus Genetic Algorithms can use such an evaluation function to design a fuzzy controller.

GAs are a new programming paradigm that has been applied to much more difficult (NP-hard) optimization problems such as scheduling with very promising results. GAs encode the solution to a given scheduling problem in a binary- (or real-valued) string (Holland, 1975, Goldberg, 1978, Michalewicz, 1994). Each string's element represents a particular feature in the solution. The string (solution) is evaluated by a fitness function to determine the solution's quality: good solutions survive and have off-springs, while bad solutions are discontinued. Solution's constraints are modeled by penalties in the fitness function or encoded directly in the solution data structures. To improve current solutions, the string is modified by two basic type of operators: cross-over and mutations. Cross-over are deterministic operators that capture the best features of two parents and pass it to a new off-spring string. Mutations are probabilistic operators that try to introduce needed solutions features in populations of solutions that lack such feature.

Some GAs have exhibited exceptional performances in large scale scheduling problems. However, many unanswered questions still remain. Design questions range from the type of solution and constraints encoding to probability of mutation, definition of fitness function, desired type of cross-over operations (to encode context dependent heuristics), etc. More fundamental questions include the applicability conditions of GAs, comparative analyses with other scheduling techniques, and, in general, a deeper understanding of the way GAs explore the solution space.

14.2 Simulated Annealing

A special case of the genetic algorithm approach is the method known as Simulated Annealing (SA), which is considered a probabilistic hill-climbing technique (Romeo and Sangiovanni-Vincentelli, 1985). SA is a more restricted version of GAs, with well understood convergence properties. Simulated Annealing can be seen as a GA in which crossovers are disabled and only mutations implemented by the probability of jumping the energy barrier are allowed. Furthermore, the population size is typically one. SA is also a global search strategy and can work in very high-dimensional searches, given enough computational resources. An interesting hybrid algorithm that spans the space from GAs to SAs has been proposed by Adler. In his algorithm the GAs operators use Simulated Annealing to determine if the newly generated solution is better than the best of its parents (in the case of the crossover operator) or better than the original solution (in the case of the mutation operator) (Adler, 1993).

15 Hybrid Algorithm: The symbiosis

Over the past few years we have seen an increasing number of hybrid algorithms, in which two or more of Soft Computing technologies (FL, NN, GA) have been integrated to improve the overall algorithm performance. In the sequel we will analyze a few of such combinations.

15.1 NN controlled by FL

Fuzzy logic enables us to easily translate our qualitative knowledge about the problem to be solved, such as resource allocation strategies, performance evaluation, and performance control, into an executable rule set. As a result, fuzzy rule bases and fuzzy algorithms have been used to monitor the performance of NNs or GAs and modify their control parameters. For instance, FL controllers have been used to control the learning rate of Neural Networks to improve the crawling behavior typically exhibited by NNs as they are getting closer to the (local) minimum. More specifically, the typical equation for the weight changes in a NN is:

$$\Delta W_n = -\eta \nabla E(W_n) + \alpha \Delta W_{n-1} \tag{25}$$

in which ΔW_n represents the changes to the weight vector W_n, $E(W_n)$ is the error function at the nth iteration, η is the learning rate and α is the momentum. The learning rate η is a function of the step size k and determines how fast the algorithm will move along the error surface, following its gradient. Therefore the choice of η has an impact on the accuracy of the final approximation and on the speed of convergence. The smaller the value of η the better the approximation but the slower the convergence. Jacobs (Jacobs, 1988) established a heuristic rule, known as the *Delta-bar-delta rule* to increase the size of η if the sign of ∇E was the same over several consecutive steps. Arabshahi et al. (Arabshahi *et al.*, 1992) developed a simple Fuzzy Logic controller to modify η as a function of the error and its derivative, considerably improving Jacobs heuristics.

15.2 GAs controlled by FL

The use of Fuzzy Logic to translate and improve heuristic rules has also been applied to manage the resource of GAs (population size, selection pressure) during their transition from *exploration* (global search in the solution space) to *exploitation* (localized search in the discovered *promising* regions of that space (Cordon *et al.*, 1995, Herrera *et al.*, 1995a, Lee and Tagaki, 1993). In (Lee and Tagaki, 1993) the authors summarize the results of Lee's Ph.D. thesis (Lee, 1994) and propose a Fuzzy controller to perform a run-time tuning of three GA parameters.

The controller takes the following three inputs to determine the current state of the GA evolution:

$$\frac{Average\,Fitness}{Best\,Fitness},\ \frac{Worst\,Fitness}{Average\,Fitness},\ \Delta Best\,Fitness$$

and produce three outputs

$$\Delta Population\,Size,\ \ \Delta Crossover\,Rate,\ \ \Delta Mutation\,Rate$$

that modify the GA parameters. These changes are constrained so that the previous values will not change by more than 50%. Furthermore the three parameters (Population Size, Crossover Rate and Mutation Rate) are limited to remain within the operational ranges [2, 160], [0.2, 1.0], and [0.0001, 1.0], respectively. Their experimental results show large improvements of computational run-time efficiency at the expense of large amounts of offline computation required to tune the Fuzzy Controller.

In general we can state that the management of GA resources gives the algorithm an adaptability that improves its efficiency and converge speed. The crucial aspect of this approach is to find the correct balance between the computational resurces allocated to the meta-reasoning (e.g. the fuzzy controller) and to the object-level problem-solving (e.g. the GA). This additional investment of resources will pay off if the controller is extendable to other object-level problem domains and if its run-time overhead is offset by the run-time performance improvement of the algorithm.

According to reference (Herrera and Lozano, 1996), this adaptability can be used in the GA's parameter settings, genetic operators selection, genetic operators behavior, solution representation, and fitness function.

In the same reference we can see two examples of this adaptability used to avoid the premature convergency of the GA to an inferior solution. This problem occurs when, due to selection pressure, disruption caused by the crossover operators, and inadequate parameter settings, the GA exhibits a lack of diversity in its population.

The first approach is based on dynamic crossover operators applied to real-coded chromosomes. These operators use different type of aggregators: *t-norms* and *t-conorms* to emphasize exploration properties, and *averaging operators* to show exploitation properties.

The second approach (in the same reference) uses two FL controllers to control the use of the exploitative crossover and the selection pressure. For this purpose two diversity measures are defined: the *genotypic diversity*, which measures the (normalized) average distance of the population from the best chromosome; and the *phenotypic diversity*, which measures the ratio between the best fitness and the average fitness. These diversity measures are the inputs to the FLCs. Every five generations the FLCs evaluate these measures to adjust the probability of using an exploitative crossover (based on averaging aggregators) and the selection pressure (keeping or eliminating diversity in the next generation).

It should be noted that there are other ways of controlling the GAs parameters setting. Specifically, GAs have also been applied at the meta-level to control the resource parameters of object-level GAs (Grefenstette, 1986).

15.3 FL Controller Tuned by GAs

Many researchers have explored the use of genetic algorithms to tune fuzzy logic controllers. Reference (Cordon *et al.*, 1995) alone contains an updated bibliography of over 300 papers combining GAs with fuzzy logic, of which at least half are specific to the tuning and design of fuzzy controllers by GAs. For brevity's sake we will limit this section to a few contributions. These methods differ mostly in the order or the selection of the various FC components that are tuned (termsets, rules, scaling factors).

One of the precursors in this quest was C. Karr (Karr, 1991b, Karr, 1991a, Karr, 1993), who used GAs to modify the membership functions in the termsets of the variables used by the FCs. Karr used a binary encoding to represent three parameters defining a membership value in each termset. The binary chromosome was the concatenation of all termsets. The fitness function was a quadratic error calculated for four randomly chosen initial conditions.

Herrera, Lozano, and Verdegay (Herrera *et al.*, 1995b) directly tuned each rule used by the FC. They used a real encoding for a four-parameter characterization of a trapezoidal membership value in each termset. Each rule was represented by the concatenation of the membership values used in the rule antecedent (state vector) and consequent (control action). The population was the concatenation of all rules so represented. A customized (max-min arithmetical) crossover operator was also proposed. The fitness function was a sum of quadratic errors.

Kinzel, Klawon and Kruse (Kinzel *et al.*, 1994) tuned both rules and termsets. They departed from the string representation and used a (cross-product) matrix to encode the rule set (as if it were in table form). They also proposed customized (point-radius) crossover operators which were similar to the two-point crossover for string encoding. They first initialized the rule base according to intuitive heuristics, used GAs to generate better rule base, and finally tuned the membership functions of the best rule base. This order of the tuning process is similar to that typically used by self-organizing controllers (Burkhardt and Bonissone, 1992b).

Lee and Takagi also tuned the rule base and the termsets (Lee and Takagi, 1993). They used a binary encoding for each three-tuple characterizing a triangular membership distribution. Each chromosome represents a Takagi-Sugeno rule(Takagi and Sugeno, 1985), concatenating the membership distributions in the rule antecedent with the polynomial coefficients of the consequent.

Also interesting is the approach taken by Surman, Kanstein, and Goser (Surmann *et al.*, 1993), who modify the usual quadratic fitness function by addition an entropy term describing the number of activated rules.

In (Bonissone *et al.*, 1996), we followed the tuning order suggested by Zheng (Zheng, 1992) for manual tuning. We began with macroscopic effects, by tuning the FC state and control variable *scaling factors*, while using a standard uniformly spread termset and a homogeneous rule base. After obtaining the best scaling factors, we proceeded to tune the *termsets*, causing medium-size effects. Finally, if additional improvements were needed, we tuned the *rule base* to achieve microscopic effects.

This parameter sensitivity order can be easily understood if we visualize a homogeneous rule base as a rule table: a modified scaling factor affects the entire rule table; a modified term in a termset affects one row, column, or diagonal in the table; a modified rule only affects one table cell.

15.4 NNs Generated by GAs

There are many forms in which GAs can be used to synthesize or tune NN: to evolve the network *topology* (number of hidden layers, hidden nodes, and number of links) letting then Back-Propagation (BP) tune the net; to find the optimal set of weights for a given topology, thus replacing BP; and to evolve the reward function, making it adaptive. The GA chromosome needed to directly encode both NN topology and parameters is usually too large to allow the GAs to perform an efficient global search. Therefore, the above approaches are usually mutually exclusive, with a few exceptions (Maniezzo, 1994, Patel and Maniezzo, 1994) that rely on variable granularity to represent the weights.

Montana and Davis were among the first to propose the use of GAs to train a feedforward NN with a given topology (Montana and Davis, 1989).

Typically NNs using Back-Propagation (BP) converge faster than GAs due to their exploitation of local knowledge. However this local search frequently causes the NNs to get stuck in a local minima. On the other hand, GAs are slower, since they perform a global search. Thus GAs perform efficient coarse-granularity search (finding the promising region where the global minimum is located) but they are very inefficient in the fine-granularity search (finding the minimum). These characteristics motivated Kitano to propose an interesting hybrid algorithm in which the GA would find a *good* parameter region which was then used to initialize the NN. At that point, Back-Propagation would perform the final parameter tuning (Kitano, 1990).

McInerney and Dhawan improved Kitano's algorithm by using the GA to escape from the local minima found by the backpropagation during the training of the NNs (rather than initializing the NNs using the GAs and then tuning it using BP). They also provided a dynamic adaptation of the NN learning rate (McInerney and Dhawan, 1993).

For an extensive review of the use of GAs in NNs, the reader is encouraged to consult references (Schaffer *et al.*, 1992) and (Yao, 1992).

15.5 FL Controller Tuned by NNs

Among the first to propose the combined use of FL and NNs was S.C. Lee (Lee and Lee, 1974), who in 1974 proposed a multi-input/multi-output neuron model, in contrast with the binary-step output function advocated in the mid seventies. Since then we have witnessed many FL-NNs combinations (see reference (Takagi, July 1990) for a more exhaustive coverage).

Within the limited scope of using NNs to tune FL Controllers, we already mentioned the seminal work on ANFIS (Adaptive Neural Fuzzy Inference Systems) by R. Jang (Jang, 1993). ANFIS consists of a six layers generalized network. The first and sixth layers correspond to the system inputs and outputs. The second layer defines the fuzzy partitions (termsets) on the input space, while the third layer performs a differentiable T-norm operation, such as the product or the soft-minimum. The fourth layer normalizes the evaluation of the left-hand-side of each rule, so that their degrees of applicability λ_i (see equation 4) will add up to one. The fifth layer computes the polynomial coefficients in the right-hand-side of each Takagi-Sugeno rule (as described in equation 2). Jang's approach is based on a two-stroke optimization process. During the forward stroke the termsets of the second layer are kept equal to their previous iteration value while the coefficients of the fifth layer are computed using a Least Mean Square method. At this point ANFIS produces an output, which is compared with the one from the training set, to produce and error. The error gradient information is then used in the backward stroke to modify the fuzzy partitions of the second layer. This process is continued until convergence is reached.

Many other variations of FLC tuning by NN have been developed, such as the ones described in references (Kawamura *et al.*, March 1992), (Bersini *et al.*, 1993), and (Bersini *et al.*, 1995),

15.6 Hybrid GAs

Since GAs are quite robust with respect to being trapped in local minima (due to the global nature of their search) but rather inaccurate and inefficient in finding the global minimum, several modifications have been proposed to exploit their advantage and compensate for their shortcoming. Of special interest is the work by Renders and Bersini, who proposed two type of hybrid GAs (Renders and Bersini, 1994). The fist type consists in interwoving GAs with Hill Climbing techniques (GA+HC): the solution selection no longer depends on the *instantaneous* evaluation of the fitness function applied to the solution but rather applied to a refinement of the solution obtained via Hill Climbing techniques. The second type of hybrid consists in embedding optimization techniques in the crossover operator used by the GAs. The population size is $\lambda(n+1)$ individuals, of which only λ individuals are replaced by offsprings. Each offspring is obtained from a group of $(n+1)$ parents via a *simplex crossover*. His results show by combining the two hybrid methods, (GA+HC) and (Simplex crossover) the resulting hybrid algorithm outperforms each of its components in achieving maximum fitness, reliably, accurately, and minimizing computing time. An analysis of the tradeoff between accuracy, reliability and computing time for hybrid GAs can be found in reference (Renders and Flasse, 1996).

16 Hybrid Systems Applications

16.1 Example of FL Controller Tuned by GAs

We will briefly summarize the transportation problem described in (Bonissone *et al.*, 1996) as a typical example of a hybrid system application. In this case study we describe the design and tuning of a controller for enforcing compliance with a prescribed velocity profile for a rail-based transportation system. This requires following a trajectory, rather than fixed setpoints (as in automobiles). We synthesize a fuzzy controller for tracking the velocity profile, while providing a smooth ride and staying within the prescribed speed limits. We use a genetic algorithm to tune the fuzzy controller's performance by adjusting its parameters (the scaling factors and the membership functions) in a sequential order of significance. We show that this approach results in a controller that is superior to the manually designed one, and with only modest computational effort. This makes it possible to customize automated tuning to a variety of different configurations of the route, the terrain, the power configuration, and the cargo. In the rest of this section we will describe the problem, our system's architecture, the tuning of the FC, and some selected results. Finally, we will conclude with an assessment of the current system and possible future extensions.

16.1.1 Problem Description

We propose a system, composed of a *cruise planning* and a *cruise control* module, that will automate the controls of a freight train. This system will be applicable during most of the train journey, except for the initial and final transients, i.e. the train starting and stopping. In this paper we will focus on using a GA to improve the performance of the on-line controller.

A freight train consists of several hundred heavy masses connected by couplers. Each coupler may have a dead zone and a hydraulically damped spring. This implies that the railcars can move relatively to each other while in motion, leading to a train that can change length by as much as 50–100 feet. Handling of the locomotive controls has a direct effect on these inter-car coupler dynamics and the forces and distances therein (which are termed *slack*). Couplers are subjected to two types of forces which may lead to breakage of the coupler, the brake pipe, and the train – *static* forces, which exceed a certain threshold, and *dynamic* forces, such as impulse impacts, which may snap the coupler. In addition, violation of speed limits and excess acceleration/breaking may also lead to derailments and severe cargo damage. Smooth handling while following a speed target is therefore absolutely imperative. Usually this handling is provided by an experienced train engineer.

Summary of Approach We will focus on the module responsible for on-line tracking of the externally supplied speed profile to drive the train smoothly and without violating the speed constraints. The profile is the reference provided according to railroad operating rules and requirements. The controller uses a fuzzy logic PI to minimize tracking error, while providing a smooth ride and insuring that no part of the train[3] will exceed the posted speed limit. The Fuzzy Proportional Integral (FPI) controller uses the tracking error and its change-in-error to recommend a change in the control output. This is used to modify the current control settings if feasible. The control outputs are the notch and brake settings (which control the acceleration of the train). If allowed, the FPI will track the reference profile accurately.

The computation of the error used by the FPI incorporates a lookahead to properly account for the train inertia. The algorithm predicts the future velocity of the train and incorporates not only the current error, but also the future predicted error, since the future reference speed is known from the profile. Finally, the PI is tuned (offline) using GA's that modify the controllers most sensitive parameters: scaling factors (SF) and membership distributions (MF).

This system has multiple purposes: providing a certain degree of train handling uniformity across all crews, enforcing railroad safety rules (such as insuring that the posted speed limits are never exceeded by any part of the train), maintaining the train schedule within small tolerances, operating the train in efficient regimes, and maintaining a smooth ride by avoiding sudden accelerations or brake applications This last constraint will minimize damage due to poor slack-handling, bunching, and run-in.

[3] It is important to remember that the typical freight train can be as long as 2.0 miles and requires up to four locomotives to pull it.

Prior Work: Problem Domain As described above, the handling of freight trains involves a multi-body problem and proper slack management, without sensors for most of the state. This leads to a much more complex problem, which cannot be solved by the simpler schemes used by the cruise controllers for other vehicles, such as cars, trucks, boats etc.

Current locomotives are equipped with a very simplistic cruise control that uses a linear Proportional Integral (PI) controller, which can be used only below speeds of 10 mph. This PI controller is primarily meant to be used for uniform loading, yard movement etc. and does not prescribe braking action. Furthermore, the technology used does not consider slack or distributed dynamics in any way, and is inappropriate for extended trains at cruising speeds over general terrain. To test and tune our fuzzy controller, we have used a simulator developed in-house, based on work done at GE and the Association of American Railroads.

16.1.2 Solution Description

System Architecture The overall scheme for the proposed train handling is shown in Figure 22. It consists of an FPI closing the loop around the train simulation (TSIM), using only the current velocity as its state input. This speed and lookahead are compared with the desired profile to generate a predicted near-term error as input to the FPI, which outputs control actions back to the simulator. Offline, a GA uses the setup for evaluating various FPI parameters.

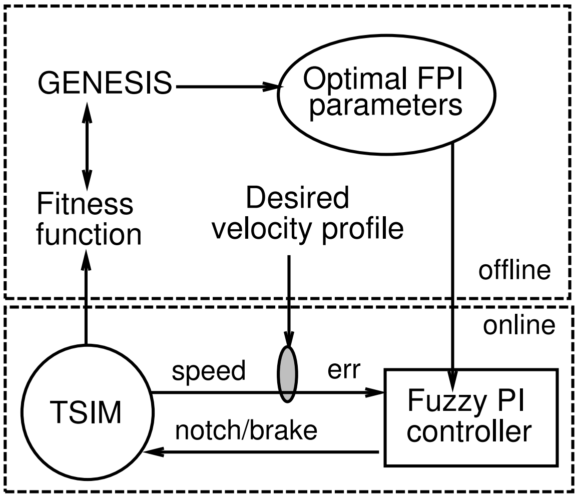

Figure 22: System schematic for using a GA to tune a fuzzy train controller.

The simulator — TSIM, is an in-house implementation, combining internal data with physical/empirical models as described in TEM (Train Energy Model) (Drish, 1992). TEM was developed by the Association of American Railroads. TSIM simulates an extended train with arbitrary detailed makeup, over a specified track profile. The version of TSIM used here does not simulate a stretchable train for the sake of computational efficiency.

For the purpose of this study, GENESIS (GENEtic Search Implementation System) has been adopted as a software development tool. It has been developed by John J. Grefenstette to promote the study of genetic algorithms for function optimization. The user must provide only a "fitness" function which returns a corresponding value for any point in the search space.

Fitness Functions To study the effects of different objectives, we minimize three fitness functions :

$$\begin{aligned} f_1 &= \sum_i |notch_i - notch_{i-1}| \\ &\quad + |brake_i - brake_{i-1}| \\ f_2 &= \sum_i |v_i - v_i^d| \\ f_3 &= w_1 \frac{\sum_i |notch_i - notch_{i-1}|}{K_1} \end{aligned}$$

$$+w_2 \frac{\sum_i |v_i - v_i^d|}{K_2}$$

v^d denotes the desired velocity and i is a distance or milepost index. f_1 captures throttle jockeying, f_2 captures speed profile tracking accuracy, and f_3 combines a weighted sum of the two.

Tuning the Fuzzy PI

Testbed and Design Choices For comparison purposes, twelve tests have been conducted by taking a cross-product of: the scaling factor values before and after GA tuning, the membership function parameter values before and after GA tuning, and the 3 fitness functions. The tests are designed to demonstrate that GAs are powerful search methods and are very suitable for automated tuning of FPI controllers. GAs are able to come up with near-optimal FPI controllers within a reasonable amount of time according to different search criteria. In addition, the tests are also designed to demonstrate that we should tune parameters in the order of their significance. That is, we should tune scaling factors first since they have global effects on all the control rules in a rule base. Tuning membership functions will only give marginal improvements for a FPI with tuned scaling factors. In the following, we present testbed set-up and design choices in brief.

• **Train Simulator Parameters:** All testing for the automated tuning of FPI was done using TSIM. Two track profiles were used: an approximately 14 mile flat track and an approximately 40 mile piece of actual track from Selkirk to Framingham over the Berkshires. The train was nearly 9000 tons, and about a mile long, with 4 locomotives and 100 loaded railcars. An analytically computed velocity profile which minimizes fuel consumption was used as the reference.

• **FPI Controller Parameters:** The standard termset used in the FPI is {NH, NM, NL, ZE, PL, PM, PH}, where N = Negative, P = Positive, H = High, M = Medium, L = Low, and ZE = Zero. Initially, these seven terms are uniformly positioned trapezoids overlapping at a 50% level over the normalized universe of discourse. This is illustrated in Figure 23. Since the controller is defined by a nonlinear control surface in

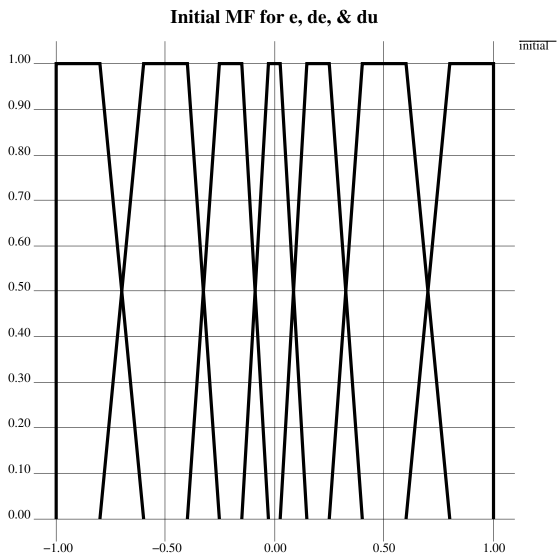

Figure 23: Fuzzy membership functions

$(e, \Delta e, \Delta u)$ space, we need three termsets in all, one for each of $e, \Delta e, \Delta u$. The design leads to a symmetric controller, which is not always a good assumption. In this case, the GA tuning will automatically create the required asymmetry.

• **GENESIS parameters:** The GA parameters are : Population Size = 50, Crossover Rate = 0.6, Mutation Rate = 0.001. In addition, all structures in each generation were evaluated, the elitist strategy was used to guarantee monotone convergence, gray codes were used in the encodings, and selection was rank based.

• **Tuning of Scaling Factors (SF)**: Each chromosome is represented as a concatenation of three 3-bit values for the three floating point values for the three FPI scaling factors S_e, S_d and S_u. They are in the ranges: $S_e \in [1,9], S_d \in [0.1, 0.9], S_u \in [1000, 9000]$. The intent is to demonstrate a GA's ability for function optimization even with such a simple 9-bit chromosome.

• **Tuning of Membership Functions (MF)**: Each MF is trapezoidal and parameterized by left base (L), center base (C), and right base (R), as illustrated in Figure 24.

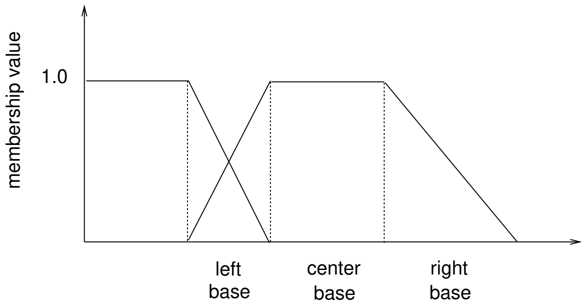

Figure 24: Membership function representation.

If we did not have any constraints on the membership functions (MFs), we would need 21 parameters to represent each termset. In our case we want to impose certain conditions on the MFs to enforce some good design rules. Since each MF is trapezoidal and we want to maintain an overlap of degree equal to 0.5 between adjacent trapezoids, this restriction partitions the universe of discourse into disjoint intervals, denoted by b_i, which alternate between being cores and the overlap areas of the MFs. Furthermore, the cores of NH and PH extend semi-infinitely to the left and right respectively (outside the [-1,1] interval). We also want to maintain a symmetry between the core of NH and PH with respect to the mid point of the scale (0). Finally, the sum of all the disjoint intervals should be equal to 2. The above constraints translate into the following relations:

- $L_i = R_{i+1}$ for $i = 1, \ldots, 6$ - [0.5 overlap]
- $R_1 = L_7 = 0$ - [Extension outside the [-1,1] interval]
- $(b_1 = C_1) = (b_{13} = C_7)$ - [Symmetry of extreme labels]
- $C_1 = \frac{2-\sum_{i=2}^{12} b_i}{2}$ - [interval normalization]

Therefore, in this case we need eleven intervals, denoted by b_i, to represent the seven 7 MF labels in each termset. In general, under the above conditions, the number of required intervals $\mid b \mid$ is:

$$\mid b \mid = (2 \times \mid MF \mid -3)$$

where $\mid MF \mid$ is the number of membership functions. Each termset is represented by a vector of 11 floating point values. For the present study, each b_i is set within the range of $[0.09, 0.18]$ and the values within this range are represented by five bits. The chromosome required to tune the membership functions is constructed by catenating the representations of the three termsets, for a total of 33 real-valued parameters.

16.1.3 Simulation Results

The discussion of the results is divided into four parts. First, we compared the simulation results between FPI with manually tuned scaling factors and that of tuned by GAs with respect to different fitness functions. These tests were corresponding to Row 1 and 2 in Table 3. Then, we compared the simulation results of different combinations of scaling factors and membership functions with respect to f_1. These tests were corresponding to Column 1 in Table 3. After that, we presented the simulation results obtained from the tests of Column 3 in Table 3. In other words, we compared the effects between tuning scaling factors and membership functions for a FPI controller with respect to f_3. We concluded this section with a summary of the simulation results.

Scaling factors	Time (min)	Journey (mile)	Fuel (gal)	Fitness
Initial [5.0,0.2,5000]	26.51	14.26	878	$f_1 = 73.2, f_2 = 227, f_3 = 1$
GA wrt. f_1 [9.0,0.1,1000]	27.76	14.21	857	$73.2 \rightarrow 15.2$
GA wrt. f_2 [6.7,0.1,4429]	25.81	14.12	875	$227 \rightarrow 213.1$
GA wrt. f_3 [3.3,0.9,5571]	27.26	14.27	875	$1.0 \rightarrow 0.82$

Table 3: Results after 4 generations with different fitness functions.

SF : Manually Tuned vs. GA The GA was started with an initial set of scaling factor values, and run with each of the fitness functions f_1, f_2, f_3 for the flat track and unit train. The results after four generations are shown in Table 3.

Throttle jockeying is almost eliminated in f_1 by decreasing both K_p and K_i simultaneously. There are no big differences in fuel consumption between the four runs. These results will show further improvement when a more dynamic fuel consumption model which models transients is incorporated into the simulator.

Tuning SF vs. MF with f_1 Four sets of tests were conducted for the comparison of significance in tuning scaling factors (SF) vs. tuning membership functions (MF) with respect to fitness function f_1. The results are shown in Table 4. The initial set of scaling factors is $[S_e S_d S_u] = [5.0, 0.2, 5000]$. After 4 generations of evolution, the set of GA tuned SF is $[9.0, 0.1, 1000]$. As shown in Table 4, the fitness value was dramatically reduced from 73.20 to 15.15 for the GA tuned SF with respect to f_1. Substantially smoother control results.

Description	Time	Journey	Fuel	Fitness	Generation
Initial SF, initial MF	26.51	14.26	878	73.20	0
GA tuned SF, initial MF	27.76	14.21	857	15.15	4
Initial SF, GA tuned MF	26.00	14.18	879	70.93	20
GA tuned SF, GA tuned MF	28.26	14.12	829	14.64	10

Table 4: Results using f_1 with different parameter sets.

On the other hand, GA tuned MF only reduced throttle jockeying a little bit after 20 generations of evolution. It is indicated by the the small reduction in fitness value from 73.20 to 70.93. From the above observations, we can conclude that tuning SF is more cost-effective than tuning MF. The little improvement by tuning MF's leads to an asymmetric shift in the membership functions, such that the ones on the left of ZE get shifted more than the ones on the right. This is due to the slightly asymmetric calibration of the notch and brake actuators. A +5 (notch) may not lead to the exact tractive force forward that a -5 (brake) leads to in the opposite direction. As a result, the best FPI responds to negative error slightly differently than to positive error.

Next, we demonstrate that tuning membership functions only gives marginal improvements for a fuzzy PI controller with tuned scaling factors. we used GA to optimize FPI's MF with respect to f_1, while using the GA tuned SF values of $[9.0, 0.1, 1000]$. After 10 generations of evolution, the fitness value was further reduced from 15.15 to 14.64. The change in smoothness and the MF values is minimal. After observing the simulation results, we conclude that tuning membership functions alone will only provide marginal improvements for a fuzzy PI controller with fine tuned scaling factors.

Tuning SF vs. MF with f_3 We further verified our arguments stated in the last sub-section with the following four sets of tests conducted with respect to fitness function f_3, which balances both tracking error and control smoothness. The results are shown in Table 5.

Recall that the initial set of scaling factors is $[5.0, 0.2, 5000]$. After 4 generations of evolution, the GA came up with a set of SF $=[3.3, 0.9, 5571]$, with respect to f_3. As shown in Table 5, the fitness value was

Description	Time	Journey	Fuel	Fitness	Generation
Initial SF w/initial MF	26.51	14.26	878	1.000	0
GA tuned SF w/initial MF	27.21	14.35	871	0.817	4
Initial SF w/GA tuned MF	26.26	14.18	871	0.942	20
GA tuned SF w/GA tuned MF	27.26	14.35	872	0.817	10

Table 5: Results using f_3 with different parameter sets.

reduced from 1.000 to 0.817 while doing this. On the other hand, GA tuned MF only reduced the fitness value from 1.000 to 0.942 after 20 generations of evolution, confirming the claim that tuning SF is more cost-effective than tuning MF.

We proceeded to experiment with MF tuning with tuned SF $= [3.3, 0.9, 5571]$. This time there were no significant improvements in fitness after 10 generations.

Description	f_1	f_2	f_3
Initial SF w/initial MF	73.2	227	1.00
GA tuned SF w/initial MF	15.2	213	0.82
Initial SF w/GA tuned MF	70.9	201	0.94
GA tuned SF w/GA tuned MF	14.6	204	0.82

Table 6: Summary of all simulation results on the flat track.

Table 6 summarizes all the 12 tests run on the flat track. In addition, we present the final performance graphs in Figure 25 for the more complex piece of real track with the same train. It shows substantial improvement in control accuracy and smoothness.

16.1.4 Application Conclusions

We have presented an approach for tuning a fuzzy KB for a complex, real-world application. In particular, we used genetic algorithms to tune scaling factors as well as membership functions, and demonstrated that the approach was able to find good solutions within a reasonable amount of time under different train handling criteria. In addition, we showed that all parameters do not need to be treated as equally important, and that sequential optimization can greatly reduce computational effort by doing scaling factors first. Additional improvement was shown by the good performance of fairly coarse encodings. The scalability of the approach enables us to customize the controller differently for each track profile, though it does not need to be changed for different train makeups. In this way, we can produce customized controllers offline efficiently. Future extensions of this system will focus on automatic generation of velocity profiles for the train simulator by using genetic algorithms for trajectory optimization subject to "soft" constraints.

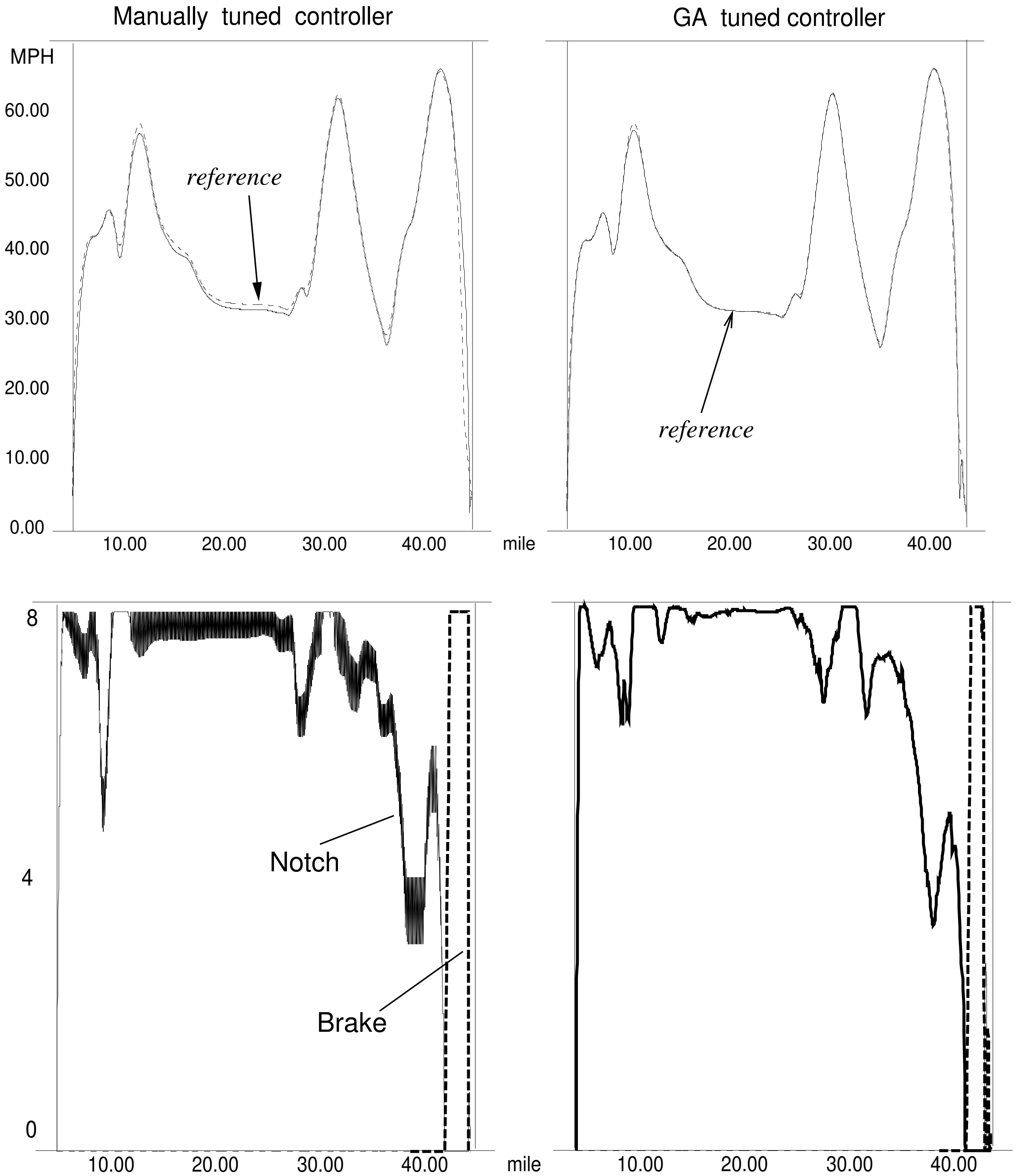

Figure 25: The two figures on the left show reference tracking performance and control outputs before GA tuning. After tuning, the two figures on the right show vastly improved tracking and smoothness of control. The track was a section of Selkirk—Framingham track, and the simulated train was the 9000 ton, mile-long, loaded unit train.

16.2 Example of NN Controlled by a FLC

This example illustrates the use of a FLC to accelerate the performance of a neural network. The original problem in which it was used was the failure prediction and failure diagnostics of a complex industrial process. The details of the problem domain are still considered confidential and cannot be divulged. However, we can certainly describe the portion concerning the acceleration of the NN-based classifier.

16.2.1 Neural Learning with Fuzzy Accelerators

The training method employed for neural nets in our approach is error back-propagation, which is a generalization of the Least Mean Squares (LMS) algorithm. This is essentially a gradient descent method over weight space, which seeks to minimize the mean squared error over the entire training set. Since gradient descent can be very slow, we need an acceleration technique to speed up neural learning. This is done using fuzzy rules.

As mentioned before, in section 15.1, the basic weight update equation in backpropagation is as follows:

$$W_{ij}^{s+1}(l) = W_{ij}^{s}(l) - \eta \frac{\partial E}{\partial W_{ij}^{s}(l)} + \alpha \left[W_{ij}^{s}(l) - W_{ij}^{s-1}(l) \right]$$

where $W_{ij}(l)$ is the weight between the ith neuron at the lth layer and the jth neuron at the $(l-1)$th layer, E is the sum of squared error between target and actual output, s is the iteration step, η is the learning rate (usually between 0.01 and 1.0), and α is the momentum coefficient (usually 0.9).

The learning rate η determines how fast the algorithm will move along the error surface, following its gradient. The momentum represents the fraction of the previous changes to the weight vector ΔW_{n-1} which will stil be used to compute the current change. As implied by its name, the momentum tends to maintain the changes moving along the same direction.

From the above equation, it is clear that the efficiency of weight updating depends on the selection of the learning rate as well as the momentum coefficient. The selection of these parameters involves a tradeoff: in general, large η and α result in fast error convergence, but poor accuracy. On the contrary, small η and α lead to better accuracy but slow training (Wasserman, 1989).

Unfortunately, the selections are mainly ad hoc, i.e., based on empirical results or trial and error. In addition, the choice of activation function, $f(x)$ could also influence the learning process:

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

where β is the steepness parameter of the activation function.

The Fuzzy Accelerator used in our application is similar to that described in (Kuo *et al.*, 1993), since η, α, and β are tuned *simultaneously*. However, the main difference is that we use both *total error* and*total training time* as fuzzy premise variables for adjusting β instead of using only total error. It is believed that total training time will provide the *annealing* effect which yields precise convergence (Rumelhart *et al.*, 1986). Fuzzy rules for adjusting of η and α are listed in Table 7, while fuzzy rules for β are shown is Table 8.

The universe of discourse of total training error is partitioned into *Small*, *Medium*, and *Big*, while the universe of discourse of change of error between two consecutive iterations is partitioned into *Negative*, *Zero*, and *Positive*. There are only nine fuzzy rules for adjustment of both the learning rate and the momentum coefficient. The point is that the training time will be reduced significantly even with such simple fuzzy rules. The obtained simulation results demonstrate the validity of the above argument.

Change of Error	*Training Error*		
	Small	*Medium*	*Big*
Negative	Very small increase	Very small increase	Small increase
Zero	No change	No change	Small increase
Positive	Small decrease	medium decrease	Large decrease

Table 7: Fuzzy rule table for $\Delta\eta$ and $\Delta\alpha$.

In Table 8, the partitioning of total training error is the same as in Table 7, while the universe of discourse of total training time is partitioned into *Short*, *Medium*, and *Long*. Again, there are nine rules for adjustment of the steepness parameter of the activation function. For instance, the fuzzy rule in row 3 column 1 is interpreted as:

IF *error* is *Small* AND *time for training* is *long*
THEN *steepness parameter* should be *Large*

Training Time	*Training Error*		
	Small	*Medium*	*Big*
Short	Medium	Small	Small
Medium	Large	Medium	Small
Long	Large	Large	Medium

Table 8: Fuzzy rule table for β.

17 PART III Conclusions: Soft Computing

We should note that Soft Computing technologies are relatively young: Neural Networks originated in 1959, Fuzzy Logic in 1965, Genetic Algorithms in 1975, and probabilistic reasoning (beside the original Bayes' rule) started in 1967 with Dempster's and the in early 80s with Pearl's work. Originally, each algorithm had well-defined labels and could usually be identified with specific scientific communities, e.g. fuzzy, probabilistic, neural, or genetic. Lately, as we improved our understanding of these algorithms' strengths and weaknesses, we began to leverage their best features and developed hybrid algorithms. Their compound labels indicate a new trend of co-existence and integration that reflects the current high degree of interaction among these scientific communities. These interactions have given birth to Soft Computing, a new field that combines the versatility of Fuzzy Logic to represent qualitative knowledge, with the data-driven efficiency of Neural Networks to provide fine-tuned adjustments via local search, with the ability of Genetic Algorithms to perform efficient coarse-granule global search. The result is the development of hybrid algorithms that are superior to each of their underlying SC components and that provide us with the better real-world problem solving tools.

References

(Adler, 1993) D. Adler. Genetic Algorithm and Simulated Annealing: A Marriage Proposal. In *1993 IEEE International Conference on Neural Networks*, pages 1104–1109, San Francisco, CA., 1993.

(Agogino and Rege, 1987) A.M. Agogino and A. Rege. IDES: Influence Diagram Based Expert System. *Mathematical Modelling*, 8:227–233, 1987.

(Arabshahi *et al.*, 1992) P. Arabshahi, J.J. Choi, R.J. Marks, and T.P. Caudell. Fuzzy Control of Backpropagation. In *First IEEE International Conference on Fuzzy Systems*, pages 967–972, San Diego, CA., 1992.

(Armor *et al.*, 1985) A. F. Armor, R. D. Hottenstine, I. A. Diaz-Tous, and N. F. Lansing. Cycling capability of supercritical turbines: A world wide assessment. In *Fossil Plant Cycling Workshop*, Miami Beach, FL, 1985.

(Badami *et al.*, 1994) V. Badami, K.H. Chiang, , P. Houpt, and P .P. Bonissone. Fuzzy logic supervisory control for steam turbine prewarming au tomation. In *Proceedings of 1994 IEEE International Conference on Fuzzy Systems*, page submitted. IEEE, June 1994.

(Bayes, 1763) T. Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 53:370–418, 1763. (Facsimile reproduction with commentary by E.C. Molina in "Facsimiles of Two Papers by Bayes" E. Deming, Washington, D.C., 1940, New York, 1963. Also reprinted with commentary by G.A. Barnard in Biometrika, 25, 293—215, 1970.).

(Bellman and Giertz, 1973) R Bellman and M. Giertz. On the Analytic Formalism of the Theory of Fuzzy Sets. *Information Science*, 5(1):149–156, 1973.

(Berenji and Khedkar, 1992) H.R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks*, 3(5), 1992.

(Berenji and Khedkar, 1993) H.R. Berenji and P. Khedkar. Clustering in product space for fuzzy inference. In *Second IEEE International conference on Fuzzy Systems*, San Francisco, CA, 1993.

(Bersini *et al.*, 1993) H. Bersini, J.P. Nordvik, and A. Bonarini. A Simple Direct Adaptive Fuzzy Controller Derived from its Neural Equivalent. In *1993 IEEE International Conference on Neural Networks*, pages 345–350, San Francisco, CA., March 1993.

(Bersini *et al.*, 1995) H. Bersini, J.P. Nordvik, and A. Bonarini. Comparing RBF and Fuzzy Inference Systems on Theoretical and Practical Basis. In *1995 International Conference on Artificial Neural Networks (ICANN95)*, pages 169–174, Paris, France, Oct. 1995.

(Bezdek and Harris, 1978) J.C. Bezdek and J.O. Harris. Fuzzy partitions and relations: An axiomatic basis for clustering. *Fuzzy Sets and Systems*, 1:112–127, 1978.

(Bezdek, 1981) J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

(Bezdek, 1994) J.C. Bezdek. Editorial: Fuzzines vs Probability - Again (! ?). *IEEE Transactions on Fuzzy Systems*, 2(1):1–3, Feb 1994.

(Bonissone and Chiang, 1993) P.P. Bonissone and K.H. Chiang. Fuzzy logic controllers: From development to deployment. In *Proceedings of 1993 IEEE Conference on Neural Networks*, pages 610–619. IEEE, March 1993.

(Bonissone and Chiang, 1995) P. Bonissone and K. Chiang. Fuzzy logic hierarchical controller for a recuperative turbosh aft engine: from mode selection to mode melding. In J. Yen, R. Langari, and L. Zadeh, editors, *Industrial Applications of Fuzzy Control and Intelligent Syste ms*. Van Nostrand Reinhold, 1995.

(Bonissone and Decker, 1986) P.P. Bonissone and K. Decker. Selecting Uncertainty Calculi and Granularity: An Experiment in Trading-off Precision and Complexity. In L. N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 217–247. North Holland, 1986.

(Bonissone and Halverson, 1990) Piero P. Bonissone and Pete C. Halverson. Time-Constrained Reasoning Under Uncertainty. *Journal of Real Time Systems*, 2:22–45, May 1990.

(Bonissone *et al.*, 1985) P. Bonissone, P. Khedkar, and M. Schutten. Fuzzy logic control of resonant converters for power supplies. In *1995 IEEE Conference on Systems, Control Applications (CCA95)*, 1985.

(Bonissone *et al.*, 1987a) P. Bonissone, S. Gans, and K. Decker. RUM: A layered architecture for reasoning with uncertainty. In *IJCAI87*, pages 891–898. AAAI, August 1987.

(Bonissone *et al.*, 1987b) P.P. Bonissone, S.S. Gans, and K.S. Decker. RUM: A Layered Architecture for Reasoning with Uncertainty. In *Proceedings 10th Intern. Joint Conf. Artificial Intelligence*, pages 891–898. AAAI, 1987.

(Bonissone *et al.*, 1995) P.P. Bonissone, V. Badami, K. Chiang, P. Khedkar, K. Marcelle, and M. Schutten. Industrial Applications of Fuzzy Logic at General Electric. *Proceedings of the IEEE*, 83(3):450–465, March 1995.

(Bonissone *et al.*, 1996) P.P. Bonissone, P. Khedkar, and Y. Chen. Genetic algorithms for automated tuning of fuzzy controllers: A transportation application. In *Fifth IEEE International conference on Fuzzy Systems*, New Orleans, Sept. 1996.

(Bonissone, 1987) P.P. Bonissone. Summarizing and Propagating Uncertain Information with Triangular Norms. *International Journal of Approximate Reasoning*, 1(1):71–101, January 1987.

(Bonissone, 1990) P. Bonissone. Now that i have a good theory of uncertainty, what else do i need? In M. Henrion, R. D. Shachter, L. Kanal, and J. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, pages 237–253. North-Holland, Amsterdam, 1990.

(Bonissone, 1991a) P.P. Bonissone. Approximate Reasoning Systems: A Personal Perspective . In *Proceedings of the American Association for Artificial Intelligence 1991 (AAAI-91)*, pages 923–929, Anaheim, CA., July 1991. AAAI.

(Bonissone, 1991b) P.P. Bonissone. A compiler for fuzzy logic controllers. In *Proceedings of the International Fuzzy Engineering Symposium '91 (IFES'91)*, pages 706–717, November 1991.

(Bonissone, 1994) P.P. Bonissone. Fuzzy logic controllers: An industrial reality. In J. Zuradqa, R. Marks, and C. Robinson, editors, *Computatioanl Intelligence: Imitating Life*, pages 316–327. IEEE Press, Piscataway, NJ, 1994.

(Bonissone, 1997) P.P. Bonissone. Soft Computing: the Convergence of Emerging Reasoning Technologies. *Journal of Soft Computing*, 1(1):to–appear, 1997.

(Bouchon-Meunier *et al.*, 1995) B. Bouchon-Meunier, R. Yager, and L. Zadeh. *Fuzzy Logic and Soft Computing*. World Scientific, Singapore, 1995.

(Burkhardt and Bonissone, 1992a) D. Burkhardt and P. Bonissone. Automated Fuzzy Knowledge Base Generation and Tuning. In *Proceedings of the IEEE International Conference on Fuzzy S ystems 1992*, pages 179–188. IEEE, March 1992.

(Burkhardt and Bonissone, 1992b) D. Burkhardt and P.P. Bonissone. Automated Fuzzy Knowledge Base Generation and Tuning. In *Proc. FIrst IEEE Int. Conf. on Fuzzy Systems 1992 (FUZZ-IEEE92)*, pages 179–188, San Diego, CA, USA, 1992. IEEE.

(Carpenter and Grossberg, 1983) A. Carpenter and S. Grossberg. A Massively parallel architecture for a self-organizing neural pattern recognitio nmachine. *Computer, Vision, Graphics, and Image Processing*, 37:54–115, 1983.

(Carpenter and Grossberg, 1987) A. Carpenter and S. Grossberg. ART 2: Self-organization of stable category recognition codes for analog output patterns. *Applied Optics*, 26(23):4919–4930, Dec. 1987.

(Carpenter and Grossberg, 1990) A. Carpenter and S. Grossberg. ART 3 Hierarchical Search: Chemical Transmitter in Self-organizing Pattern Recognition Architectures. In *Int. Joint Conf. on Neural Networks (IJCNN'90)*, pages 30–33, Washington, DC, 1990.

(Cordon *et al.*, 1995) O. Cordon, H. Herrera, and M. Lozano. A classified review on the combination fuzzy logic-genetic algorithms bibliography. Technical report 95129, url:http://decsai.ugr.s/ herrera/flga.html, Department of Computer Science and AI, Universidad de Granada, Granada, Spain, 1995.

(DeFinetti, 1937) B. DeFinetti. La prévision: ses lois logiques, ses sources subjectives. *Annales de l'Institut H. Poincaré*, 7:1–68, 1937.

(Dempster, 1967) A.P. Dempster. Upper and lower probabilities induced by a multivalued mapping. *Annals of Mathematical Statistics*, 38:325–339, 1967.

(Di Nola and Gerla, 1986) A. Di Nola and G. Gerla. Non-standard fuzzy sets. *Fuzzy Sets and Systems*, 18:173–181, 1986.

(Drish, 1992) Jr. Drish, W. D. Train Energy Model – Technical Manual. Technical Report SD-040, Association of American Railroads (AAR), March 1992.

(Dubois and Prade, 1984) D. Dubois and H. Prade. Criteria Aggregation and Ranking of Alternatives in the Framework of Fuzzy Set Theory. In H.J. Zimmerman, L.A. Zadeh, and B.R. Gaines, editors, *TIMS/Studies in the Management Science, Vol. 20*, pages 209–240. Elsevier Science Publishers, 1984.

(Dubois and Prade, 1992) D. Dubois and H. Prade. Possibility theory as a basis for preference propagation in automated reasoning. In *First IEEE International Conference on Fuzzy Systems*, pages 821–832, San Diego, CA., March 1992.

(Dubois and Prade, 1993) D. Dubois and H. Prade. Fuzzy Sets and Probability: Misunderstandings, Bridges and Gaps. In *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, pages 1059–1068, San Francisco, CA, March 1993. IEEE.

(Duda *et al.*, 1976) R.O. Duda, P.E. Hart, and N.J. Nilsson. Subjective Bayesian methods for rule-based inference systems. In *Proc. AFIPS 45*, pages 1075–1082, New York, 1976. AFIPS Press.

(et al., 1988) K. Aoyogi et al. An expert system for startup scheduling and operation support i n fossil power plants. In *Proceedings of the International Workshop on AI for Industr ial Applications: IEEE AI '88*, pages 164–172, May 1988.

(Fagin and Halpern, 1989) R. Fagin and J. H. Halpern. Uncertainty, belief, and probability. In *Proc. 11th Intern. Joint Conf. on Artificial Intelligence*, pages 1161–1167, Detroit, MI, 1989.

(Fahlman, 1988) S. Fahlman. Faster-learning variations on backpropagation: An empirical study. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the Connectionist Model Summer School.* Morgan Kaufmann, San Mateo, CA, 1988.

(Fogel *et al.*, 1966) L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley, New York, NY, 1966.

(Fogel, 1962) L.J. Fogel. Autonomous Automata. *Industrial Research*, 4:14–19, 1962.

(Fogel, 1995) D.B. Fogel. *Evolutionary Computation.* IEEE Press, New York, NY, 1995.

(Forgy, 1982) Charles L. Forgy. Rete: A fast Algorithm for the many pattern/many object patter n match problem. *Journal of Artificial Intelligence*, 19(1):17–37, September 1982.

(Garcia and Morshedi, 1986) Carlos E. Garcia and A. M. Morshedi. Quadratic programming solution of dynamic matrix control (QDMC). *Chem. Eng. Commun.*, 46:073–087, 1986.

(Giles, 1981) R. Giles. Lukasiewicz logic and fuzzy set theory. In E.H. Mamdani and B. Gaines, editors, *Fuzzy Reasoning and its applications*, pages 117–131. Academic Press, London, 1981.

(Goldberg, 1978) D.E. Goldberg. *Genetic Algorithms.* Addison Wesly, Reading, MA., 1978.

(Grefenstette, 1986) J. Grefenstette. Optimization of Control Parameters for Genetics Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16:122–128, 1986.

(Halpern and Fagin, 1990) J. Y. Halpern and R. Fagin. Two views of belief: Belief as generalized probability and belief as evidence. In *Proc. Eight National Conference on Artificial Intelligence*, pages 112–119, Boston, MA., 1990.

(Hanzalek and Ipsen, November 7 11 1965) F. J. Hanzalek and P. G. Ipsen. Boiler-turbine coordination during start-up and loading of large units. In *American Society of Mechanical Engineers*, Chicago, IL, November 7–11, 1965.

(Henrion, 1989) M. Henrion. Practical Issues in Constructing a Bayes' Belief Network. In L.N. Kanal, T.S. Levitt, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*, pages 161–173. North-Holland, 1989.

(Herrera and Lozano, 1996) F. Herrera and M. Lozano. Adaptive Genetic Algorithms Based on Fuzzy Techniques. In *Proc. of IPMU'96*, pages 775–780, Granada, Spain, 1996.

(Herrera *et al.*, 1995a) F. Herrera, M. Lozano, and J.L. Verdegay. Tackling fuzzy genetic algorithms. In G. Winter, J. Periaux, M. Galan, and P. Cuesta, editors, *Genetic algorithms in Engineering and Computer Science*, pages 167–189. Wyley and Sons, 1995.

(Herrera *et al.*, 1995b) F. Herrera, M. Lozano, and L. Verdegay. Tuning fuzzy logic control by genetic algorithms. *Int. Journal Approximate Reasoning (IJAR)*, 12(3/4):299–315, 1995.

(Holland, 1975) J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA., 1975.

(Hopfield, 1982) J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. Natl. Acad. Sci.*, 79:2554–2558, Apr. 1982.

(Hornick *et al.*, 1989) K. Hornick, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

(Horvitz *et al.*, 1989) E. J. Horvitz, H. J. Suermondt, and G. F. Cooper. Bounded conditioning flexible inference for decisions under scarce resources. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, pages 182–193, August 1989.

(Howard and Matheson, 1984) R.A. Howard and J.E. Matheson. Influence diagrams. In R.A. Howard and J.E. Matheson, editors, *The Principles and Applications of Decision Analysis*, volume 2, pages 719–762. Strategic Decisions Group, Menlo Park, California, 1984.

(Jacobs, 1988) R.A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

(Jang *et al.*, 1993) R. Jang, J. Sun, C.-T., and C. Darken. Functional Equivalence Between Radial Basis Function Networks and Fuzzy Inference Systems. *IEEE Transactions on Neural Networks*, 4(1):156–159, 1993.

(Jang, 1993) J.S.R. Jang. ANFIS: Adaptive-network-based-fuzzy-inference-system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, May/June 1993.

(Karr, 1991a) C.L. Karr. Design of an adative fuzzy logic controller using genetic algorithms. In *Proc. Int. Conf. on Genetic Algorithms (ICGA'91)*, pages 450–456, San Diego, CA., USA, 1991.

(Karr, 1991b) C.L. Karr. Genetic algorithms for fuzzy controllers. *AI Expert*, 6(2):27–33, 1991.

(Karr, 1993) C.L. Karr. Fuzzy control of ph using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 1:46–53, 1993.

(Kawamura *et al.*, March 1992) A. Kawamura, N. Watanabe, H. Okada, and K. Asakawa. An Prototype of Neuro-Fuzzy Cooperation Systems. In *First IEEE International Conference on Fuzzy Systems*, pages 1275–1282, San Diego, CA., March 1992.

(Khedkar, 1993) P. S. Khedkar. *Learning as Adaptive Interpolation in Neural Fuzzy Systems.* PhD thesis, University of California, Berkeley, May 1993.

(Khotanzad *et al.*, February 1993) A. Khotanzad, R.C. Hwang, and D. Maratukulam. Hourly load forecasting by neural networks. In *IEEE Power Engineering Society Winter Meeting*, Columbus, OH, February 1993.

(Kickert and Mamdani, 1978) W.J.M. Kickert and E. H. Mamdani. Analysis of a fuzzy logic controller. *Fuzzy Set and Systems*, 12:29–44, 1978.

(Kim and Pearl, 1983) J. H. Kim and J. Pearl. A computational model for causal and diagnostic reasoning in inference engines. In *Proc. 8th. Intern. Joint Conf. on Artificial Intelligence*, pages 190–193, Karlsruhe, Germany, 1983.

(King and Stuart, 1983) R. King and T. Stuart. Inherent Overload Protection for the Series Resonant Converter. *IEEE Transactions on Aerospace Electronic Systems*, AES-19(6):820–830, 1983.

(Kinzel *et al.*, 1994) J Kinzel, F. Klawoon, and R. Kruse. Modifications of genetic algorithms for designing and optimizing fuzzy controllers. In *Proc. First IEEE Conf. on Evolutionary Computing (ICEC'94)*, pages 28–33, Orlando, FL., USA, 1994.

(Kitano, 1990) H. Kitano. Empirical Studies on the Speed of Convergence of Neural Networks Training Using Genetic Algorithms. In *Eigth National Conference in Artificial Intelligence (AAAI'90)*, pages 789–796, Boston, MA., 1990.

(Klement, 1981) P. Klement. Operations on Fuzzy Sets and Fuzzy Numbers Related to Triangular Norms. In *Proceedings of the 11th International Symposium on Multiple Valued Logic*, pages 218–225, Oklahoma City, OK, May 1981. IEEE Computer Society Press.

(Klir and Folger, 1988) G. Klir and T.A. Folger. *Fuzzy Sets, Uncertainty, and Information.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

(Kohonen, 1982) T. Kohonen. Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43:59–69, 1982.

(Koza, 1992) J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

(Kuo *et al.*, 1993) R. J. Kuo, Y. T. Chen, P. H. Cohen, and S. Kumara. Fast convergence of error back propagation algorithm through fuzzy modeling. In *Intelligent Engineering Systems throguh Artificial Neural Net works*, pages 239–244, St. Louis, Missouri, November 1993.

(Lauritzen and Spiegelhalter, 1988) S.L. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Roy. Stat. Soc. Ser. B*, 50, 1988.

(Lee and Lee, 1974) S.C. Lee and E.T. Lee. Fuzzy Sets and Neural Networks. *Journal of Cybernetics*, 4(2):83–103, 1974.

(Lee and Tagaki, 1993) M.A. Lee and H. Tagaki. Dynamic control of genetic algorithm using fuzzy logic techniques. In S. Forrest, editor, *Proceedings of the fifth International Conference on Genetic Algorithms*, pages 76–83. Morgan Kaufmann, 1993.

(Lee and Takagi, 1993) M. Lee and H. Takagi. Integrating design stages of fuzzy systems using genetic algorithms. In *Proc. Second IEEE Conf. on Fuzzy Systems (FUZZ-IEEE'93)*, pages 612–617, San Francisco, CA., USA, 1993.

(Lee, 1990a) C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller – Part I and II. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):404–435, 1990.

(Lee, 1990b) C.C. Lee. Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I, II. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):404–435, 1990.

(Lee, 1994) M.A. Lee. *Automatic Design and Adaptation of Fuzzy Systems and Genetic Algorithms Using Soft Computing Techniques.* PhD thesis, University of California, Davis, 1994.

(Lowrance *et al.*, 1986) J.D. Lowrance, T.D. Garvey, and T.M. Strat. A Framework for Evidential-Reasoning Systems. In *Proc. 5th National Conference on Artificial Intelligence*, pages 896–903, Menlo Park, CA., 1986. AAAI.

(Lukasiewicz, 1967) H. Lukasiewicz. Many-valued systems for propositional logic. In S. McCall, editor, *Polish Logic: 1920-1939*. Oxford University Press, 1967.

(Mamdani and Assilian, 1975) E.H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *Int. J. Man Machine Studies*, 7(1):1–13, 1975.

(Maniezzo, 1994) V. Maniezzo. Genetic Evolution of the Topology and Weight Distribution of Neural Networks. *IEEE Transactions on Neural Networks*, 1994.

(Marcelle *et al.*, 1994) K. Marcelle, K.H. Chiang, P.P. Bonissone, and P . Houpt. Optimal load cycling of large steam turbines. In *Proceedings of 1994 IEEE International Conference on Fuzzy Systems*, page submitted. IEEE, June 1994.

(Martin and G. J. Silvestri, May 10 12 1988) H. F. Martin and Jr. G. J. Silvestri. Turbine cycle performance optimization. In *EPRI Heat Rate Improvement Conference*, Richmond, VA, May 10–12, 1988.

(McInerney and Dhawan, 1993) M. McInerney and A.P. Dhawan. Use of Genetic Algorithms with Backpropagation in Training of Feedforward Neural Networks. In *1993 IEEE International Conference on Neural Networks*, pages 203–208, San Francisco, CA., March 1993.

(Michalewicz, 1994) Z. Michalewicz. *Genetic Algorithms + Data Structures = Evoultion Programs*. Springer-Verlag, New York, NY, 1994.

(Minsky and Papert, 1969) M.L. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA., 1969.

(Mizumoto and Zimmerman, 1982) M. Mizumoto and H. Zimmerman. Comparison of Various Fuzzy Reasoning Methods. *Fuzzy Sets and Systems*, 8:253–283, 1982.

(Mizumoto, 1989) M. Mizumoto. Improvements Methods of Fuzzy Controls. In *Proceedings of the Third International Fuzzy Systems Associ ation*, pages 60–62. IFSA, August 1989.

(Moller, 1990) M. Moller. *A scaled conjugate gradient algorithm for fast supervised learning.* PhD thesis, Comp. Science Dept., University of Aarhus, Denmark, 1990.

(Montana and Davis, 1989) D.J. Montana and L. Davis. Training Feedforward Neural Networks Using Genetic Algorithms. In *Eleventh International Joint Conference on Artificial Intelligence*, pages 762–767, Detroit, MI., 1989.

(Moody and Darken, 1989) J. Moody and C. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1:281–294, 1989.

(Mundici, 1995) D. Mundici. Averaging the Truth-Value in Lukaasiewicz Logic. *Studia Logica*, 55:113–127, 1995.

(Oruganti and Lee, 1984) R. Oruganti and F. Lee. Resonant power processors, part II - methods of control. In *IEEE Industrial Applications Society*, pages 1461–1471, 1984.

(Palmer, 1991) R. Palmer. Sliding Mode Fuzzy Control. In *Proceedings of the IEEE International Conference on Fuzzy S ystems 1992*, pages 519–526. IEEE, March 1991.

(Patel and Maniezzo, 1994) M.J. Patel and V. Maniezzo. NN's and GA's: Evolving co-operative Behavior in adaptive learning agents. In *First IEEE Conference on Evolutionary Computation*, pages 290–295, 1994.

(Pearl, 1982) J. Pearl. Reverend Bayes on Inference Engines: a Distributed Hierarchical Approach. In *Proceedings Second National Conference on Artificial Intelligence*, pages 133–136. AAAI, August 1982.

(Pearl, 1986a) J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

(Pearl, 1986b) J. Pearl. On evidential reasoning on a hierarchy of hypothesis. *Artificial Intelligence*, 28:9–15, 1986.

(Pearl, 1988a) J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan-Kaufmann, San Mateo, California, 1988.

(Pearl, 1988b) Judea Pearl. Evidential reasoning under uncertainty. In Howard E. Shrobe, editor, *Exploring Artificial Intelligence*, pages 381–418. Morgan Kaufmann, San Mateo, CA, 1988.

(Procyk and Mamdani, 1979) T.J. Procyk and E.H. Mamdani. A linguistic self-organizing process controller. *Automatica*, 15(1):15–30, 1979.

(Ranganathan *et al.*, 1982) V. Ranganathan, P. Ziogas, and V. Stefanovic. A regulated dc-dc voltage source converter using a high frequency link. In *IEEE Industrial Applications Society*, pages 279–287, 1982.

(Rechenberg, 1965) I. Rechenberg. Cybernetic Solution Path of an Experimental Problem. *Royal Aircraft Estaablishment, Library Translation No. 1122*, 1965.

(Renders and Bersini, 1994) J.M. Renders and H. Bersini. Hybridizing Genetic Algorithms with Hill Climbing Methods for Global Optimization: Two Possible Ways. In *First IEEE Conference on Evolutionary Computation*, pages 312–317, 1994.

(Renders and Flasse, 1996) J.M. Renders and S.P. Flasse. Hybrid Methods Using Genetic Algorithms for Global Optimization. *IEEE Trans. on Systems, Man, and Cybernetics*, 26(2):243–258, 1996.

(Romeo and Sangiovanni-Vincentelli, 1985) F. Romeo and A. Sangiovanni-Vincentelli. *Probabilistic Hill-Climbing Algorithms: Properties and Applications.* Computer Science Press, Chapel Hill, NC., 1985.

(Rosenbaltt, 1959) F. Rosenbaltt. Two theorems of statistical separability in the perceptron. In *Mechanization of Thought Processes*, pages 421–456, Symposium held at the National Physical Laboratory, HM Stationary Office, London, 1959.

(Rumelhart *et al.*, 1986) D. E. Rumelhart, J. L. McClelland, and G. E. Hinton. *Parallel Distributed Processing, Vol. 1 and 2.* The MIT Press, Cambridge, MA, 1986.

(Ruspini, 1987) E.H. Ruspini. Epistemic logic, probability, and the calculus of evidence. In *Proc. Tenth Intern. Joint Conf. on Artificial Intelligence*, Milan, Italy, 1987.

(Ruspini, 1989) E.H. Ruspini. On the Semantics of Fuzzy Logic. Technical Note 475, Artificial Intelligence Center, SRI International, Menlo Park, California, 1989.

(Ruspini, 1990) E.H. Ruspini. Possibility as similarity: The semantics of fuzzy logic. In *Proc. of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 281–289, Cambridge, MA, 1990.

(Sanders *et al.*, 1989) S. Sanders, J. Noworolski, X. Liu, and G. Verghese. Generalized Averaging Method for Power Conversion Circuits. In *IEEE Power Electronics Specialists Conference*, pages 273–290, 1989.

(Schachter, 1986) R.D. Schachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.

(Schaffer *et al.*, 1992) J.D. Schaffer, D. Whitley, and L.J. Eshelman. Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art. In *1992 International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 1–37, 1992.

(Schwarz, 1975) F. Schwarz. An Improved Method of Resonant Current Pulse Modulation for Power Converters. In *IEEE Power Electronics Specialists Conference*, pages 194–204, 1975.

(Schwefel, 1965) H.-P. Schwefel. *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik.* PhD thesis, Diploma Thesis Technical University of Berlin, Germany, 1965.

(Schweizer and Sklar, 1963) B. Schweizer and A. Sklar. Associative functions and abstract semi-groups. *Publicationes Mathematicae Debrecen*, 10:69–81, 1963.

(Schweizer and Sklar, 1983) B. Schweizer and A. Sklar. *Probabilistic Metric Spaces.* North Holland, New York, 1983.

(Shafer, 1976) G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, Princeton, NJ., 1976.

(Shafer, 1990) G. Shafer. Perspectives on the theory and practice of belief functions. *International Journal of Approximate Reasoning*, 4(5/6):323–362, 1990.

(Shingu and Nishimori, 1989) T. Shingu and E. Nishimori. Fuzzy based automatic focusing system for compact camera. In *Proceedings of the Third International Fuzzy Systems Associ ation*, pages 436–439. IFSA, August 1989.

(Shortliffe and Buchanan, 1975) E.H. Shortliffe and B. Buchanan. A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23:351–379, 1975.

(Slotine and Li, 1991) J. E. Slotine and Weiping Li. *Applied Nonlinear Control.* Prentice Hall, Englewood Cliffs, 1991.

(Smets, 1981) P. Smets. The Degree of Belief in a Fuzzy Set. *Information Science*, 25:1–19, 1981.

(Smets, 1988) P. Smets. Belief functions. In P. Smets, A. Mamdani, D. Dubois, and H. Prade, editors, *Non-Standard Logics for Automated Reasoning.* Academic Press, New York, 1988.

(Smets, 1991) Ph. Smets. The transferable belief model and other interpretations of dempster-shafer's model. In P.P. Bonissone, M. Henrion, L N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 6*, pages 375–383. North-Holland, Amsterdam, 1991.

(Smith and Comer, 1991) S. M. Smith and D. J. Comer. An Algorithm for Automated Fuzzy Logic Controller Tuning. In *Proceedings of the IEEE International Conference on Fuzzy S ystems 1992*, pages 615–621. IEEE, March 1991.

(Stark *et al.*, 1962) L. Stark, M. Okajima, and G Whipple. Computer Pattern Recognition Techniques: Electrocardiographic Diagnosis. *Comm. of the ACM*, 5:527–532, 1962.

(Steigerwald, 1987) R. Steigerwald. A comparison of half-bridge resonant converter topologies. In *IEEE Applied Power Electronics Conference*, pages 135–144, 1987.

(Stillman, 1991) J. Stillman. On heuristics for finding loop cutsets in multiply-connected belief networks. In P.P. Bonissone, M. Henrion, L N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 6*, pages 233–243. North-Holland, Amsterdam, 1991.

(Suermondt *et al.*, 1991) J. Suermondt, G. Cooper, and D. Heckerman. A combination of cutset conditioning with clique-tree propagation in the pathfinder system. In P.P. Bonissone, M. Henrion, L N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 6*, pages 245–253. North-Holland, Amsterdam, 1991.

(Sugeno *et al.*, 1991) Michio Sugeno, Toshiaki Murofushi, Junji Nishino, and Hid eaki Miwa. Helicopter flight control based on fuzzy logic. In *Proceedings of the International Fuzzy Engineering Symposiu m '91 (IFES'91)*, pages 1120–1121, November 1991.

(Sugeno, 1985) M. Sugeno, editor. *Industrial Applications of Fuzzy Control.* North Holland, Amsterdam, 1985.

(Surmann *et al.*, 1993) H. Surmann, A. Kanstein, and K. Goser. Self-organizing and genetic algorithms for an automatic design of fuzzy control and decision systems. In *Proc. EUFIT'93*, pages 1097–1104, Aachen, Germany, 1993.

(Takagi and Sugeno, 1985) T. Takagi and M. Sugeno. Fuzzy Identification of Systems and Its Applications to Modeling and Control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(1):116–132, 1985.

(Takagi, July 1990) H. Takagi. Fusion Technology of Fuzzy Theory and Neural Networks - Survey and Future Directions. In *IIZUKA'90: International Conference on Fuzzy Logic and Neural Networks*, pages 13–26, Iizuka, Japan, July 1990.

(Takahashi *et al.*, 1991) H. Takahashi, K. Ikeura, and T. Yamamori. 5-speed automatic transmission installed fuzzy reasoning. In *Proceedings of the International Fuzzy Engineering Symposiu m '91 (IFES'91)*, pages 1136–1137, November 1991.

(Tang and Mulholland, 1987) K. L. Tang and R. J. Mulholland. Comparing Fuzzy Logic with Classical Controller Design. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(6):1085–1087, 1987.

(Tollenaere, 1990) T. Tollenaere. SuperSAB: fast adaptive backpropagaation with good scaling properties. *Neural Networks*, 3:561–573, 1990.

(Trillas and Valverde, 1985) E. Trillas and L. Valverde. On Mode and Implication in Approximate Reasoning. In M. Gupta, Randel, and W. Bandler, editors, *Approximate Reasoning in Expert Systems*, pages 157–166. Elsevier, Amsterdam, The Netherlands, 1985.

(Wasserman, 1989) P. D. Wasserman. *Neural Computing: Theory and Practice.* Van Nostrand Reinhold, 1989.

(Werbos, 1974) P. Werbos. *Beyond Regression: New Tools for Predictions and Analysis in the Behavioral Science.* PhD thesis, Harvard University, Cambridge, MA, 1974.

(Widrow and Hoff, 1960) B. Widrow and M.E. Hoff. Adaptive switching circuits. In *IRE WesternElectric Show and Convention Record, Part 4*, pages 96–104, August 1960.

(Widrow, 990) B. Widrow. ADALINE: An Engineering Model for Neurocomputing. In *IIZUKA'90: International Conference on Fuzzy Logic and Neural Networks*, pages 5–9, Iizuka, Japan, July 990.

(Yao, 1992) X. Yao. A review of evolutionary artificial neuralnetworks. Technical report, Commonwealth Scientific and Industrial Research Organization, Victoria, Australia, 1992.

(Yasunobu and Miyamoto, 1985) S. Yasunobu and S. Miyamoto. Automatic train operation by predictive fuzzy control. In M. Sugeno, editor, *Industrial Applications of Fuzzy Control.* North Holland, Amsterdam, 1985.

(Zadeh, 1965) L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

(Zadeh, 1968) L.A. Zadeh. Probability Measures of Fuzzy Events. *J. Math. Analysis and Appl.*, 10:421–427, 1968.

(Zadeh, 1973) L.A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Trans. Systems, Man and Cybernetics*, SMC-3:28–44, 1973.

(Zadeh, 1978) L.A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

(Zadeh, 1979) L.A. Zadeh. A theory of approximate reasoning. In P. Hayes, D. Michie, and L.I. Mikulich, editors, *Machine Intelligence*, pages 149–194. Halstead Press, New York, 1979.

(Zadeh, 1994) L.A. Zadeh. Fuzzy Logic and Soft Computing: Issues, Contentions and Perspectives. In *IIZUKA'94: 3rd International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, pages 1–2, Iizuka, Japan, 1994.

(Zhe *et al.*, 1993) S.Z. Zhe, S.H. Tan, F.L. Xu, and P.Z. Wang. PID self-tuning control using a fuzzy adaptive mechanism. In *Proceedings of the IEEE International Conference on Fuzzy S ystems 1993*, pages 708–713. IEEE, March 1993.

(Zheng, 1992) Li Zheng. A Practical Guide to Tune Proportional and Integral (PI) Like Fuzzy Controllers. In *Proceedings of the IEEE International Conference on Fuzzy S ystems 1992*, pages 633–640. IEEE, March 1992.