

# Hybrid Soft Computing Systems: Industrial and Commercial Applications

Piero P. Bonissone, Yu-To Chen, Kai Goebel, and Pratap S. Khedkar

GE Corporate Research and Development  
One Research Circle, Niskayuna, NY 12309, USA  
{bonissone, chen, goebelk, khedkar}@crd.ge.com

## Abstract

Soft computing (SC) is an association of computing methodologies that includes as its principal members fuzzy logic, neuro-computing, evolutionary computing and probabilistic computing. We present a collection of methods and tools that can be used to perform diagnostics, estimation, and control. These tools are a great match for real-world applications that are characterized by imprecise, uncertain data, and incomplete domain knowledge. We outline the advantages of applying SC techniques and in particular the synergy derived from the use of hybrid SC systems. We illustrate some combinations of hybrid SC systems, such as fuzzy logic controllers (FLCs) tuned by neural networks (NNs) and evolutionary computing (EC), NNs tuned by EC or FLCs, and EC controlled by FLCs. We discuss three successful real-world examples of SC applications to industrial equipment diagnostics, freight train control, and residential property valuation.

## 1. Introduction

### 1.1 Motivation

There is a wide gamut of industrial and commercial problems that require the analysis of uncertain and imprecise information. Usually, an incomplete understanding of the problem domain further compounds the problem of generating models used to explain past behaviors or predict future ones. These problems present a great opportunity for the application of soft computing technologies.

In the industrial world, it is increasingly common for companies to provide diagnostics and prognostics services for expensive machinery (e.g., locomotives, aircraft engines, medical equipment). Many manufacturing companies are trying to shift their operations to the service field where they expect to find higher margins. One embodiment of this trend is the successful offering of long-term service agreements with guaranteed up time. These contracts provide economic incentives for the service provider to keep the equipment in working order. Since performing unnecessary maintenance actions is undesired and costly, in the interest of maximizing the margins, service should only be performed when required. This can be accomplished by using a tool that measures the state of the system and indicates any incipient failures. Such a tool must have a high level of sophistication that incorporates

monitoring, fault detection, decision making about possible preventive or corrective action, and monitoring of its execution. A second industrial challenge is to provide intelligent automated controllers for complex dynamic systems that are currently controlled by human operators. For instance automatic freight train handling is a task that still eludes full automation.

In the commercial and financial world, we can still find inefficient markets in which timely asset valuation and pricing still represent a big challenge. For instance, while we can easily determine the value of a portfolio of stocks and bonds, we cannot perform the same precise valuation for a portfolio of real-estate assets, which could have been used as collateral for mortgage-backed securities.

Due to of the complexity of these tasks, artificial intelligence (AI) and in particular soft computing have been called upon to help. Recently we have applied soft computing techniques in support of service tasks such as anomaly detection and identification, diagnostics, prognostics, estimation and control [1,2], and residential property valuation [3]. This paper focuses on the use of soft computing on these three aspects: diagnostics, control, and valuation. To begin, we will present our interpretation of the definition and scope of the concept of soft computing.

### 1.2 Soft Computing

Soft computing (SC) is a term originally coined by Zadeh [4] to denote systems that "... exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution cost, and better rapport with reality." Soft computing is "an association of computing methodologies that includes as its principal members fuzzy logic (FL), neuro-computing (NC), evolutionary computing (EC) and probabilistic computing (PC)" [5]. However, it should be noted that we have not reached a consensus yet as to the exact scope or definition of SC [6].

The main reason for the popularity of soft computing is the *synergy* derived from its components. SC's main characteristic is its intrinsic capability to create *hybrid systems* that are based on a (loose or tight) integration of constituent technologies. This integration provides complementary reasoning and searching methods that allow

us to combine domain knowledge and empirical data to develop flexible computing tools and solve complex problems. Figure 1 illustrates a taxonomy of these hybrid algorithms and their components. Extensive coverage of this topic can be found in [7, 8].

### 1.3 SC Components and Taxonomy

#### Fuzzy Computing

The treatment of imprecision and vagueness can be traced back to the work of Post, Kleene, and Lukasiewicz, multiple-valued logicians who in the early 1930's proposed the use of three-valued logic systems (later followed by infinite-valued logic) to represent *undetermined, unknown*, or other possible intermediate truth-values between the classical Boolean *true* and *false* values [9]. In 1937, the

stemming from the representation of sets with ill-defined boundaries; a *relational* facet, focused on the representation and use of fuzzy relations; and an *epistemic* facet, covering the use of FL to fuzzy knowledge based systems and data bases. A comprehensive review of fuzzy logic and fuzzy computing can be found in [14].

Fuzzy logic gives us a language, with syntax and local semantics, in which we can translate qualitative knowledge about the problem to be solved. In particular, FL allows us to use linguistic variables to model dynamic systems. These variables take fuzzy values that are characterized by a label (a sentence generated from the syntax) and a meaning (a membership function determined by a local semantic procedure). The meaning of a linguistic variable may be interpreted as an elastic constraint on its value. These

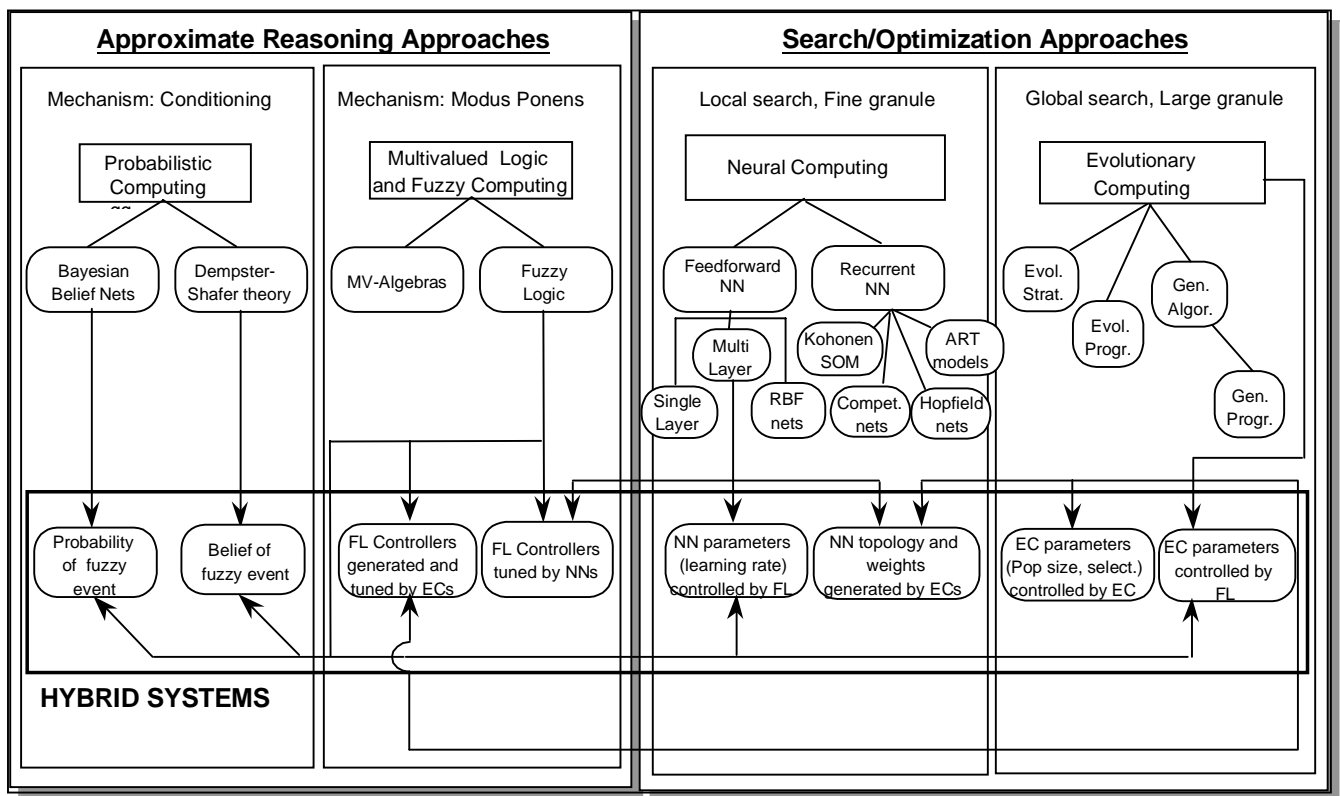


Figure 1: Soft Computing Components and Hybrid Systems

philosopher Max Black suggested the use of a *consistency profile* to represent vague concepts [10]. While vagueness relates to ambiguity, fuzziness addresses the lack of sharp set-boundaries. It was not until 1965, when Zadeh proposed a complete theory of fuzzy sets (and its isomorphic fuzzy logic), that we were able to represent and manipulate ill-defined concepts [11].

In a narrow sense, fuzzy logic could be considered a fuzzification of Lukasiewicz Aleph-1 multiple-valued logic [12]. In the broader sense, however, this narrow interpretation represents only one of FL's four facets [13]. More specifically, FL has a *logical* facet, derived from its multiple-valued logic genealogy; a *set-theoretic* facet,

constraints are propagated by fuzzy inference operations, based on the *generalized modus-ponens*. This reasoning mechanism, with its interpolation properties, gives FL a robustness with respect to variations in the system's parameters, disturbances, etc., which is one of FL's main characteristics [15].

#### Probabilistic Computing

Rather than retracing the history of probability, we will focus on the development of probabilistic computing (PC) and illustrate the way it complements fuzzy computing. As depicted in Figure 1, we can divide probabilistic computing into two classes: single-valued and interval-valued systems.

Bayesian belief networks (BBNs), based on the original work of Bayes [16], are a typical example of single-valued probabilistic reasoning systems. They started with approximate methods used in first-generation expert systems, such as MYCIN's confirmation theory [17] and PROSPECTOR's modified Bayesian rule [18], and evolved into formal methods for propagating probability values over networks [19-20]. In general, probabilistic reasoning systems have exponential complexity, when we need to compute the joint probability distributions for *all* the variables used in a model. Before the advent of BBNs, it was customary to avoid such computational problems by making unrealistic, global assumptions of conditional independence. By using BBNs we can decrease this complexity by encoding domain knowledge as structural information: the presence or lack of conditional dependency between two variables is indicated by the presence or lack of a link connecting the nodes representing such variables in the network topology. For specialized topologies (trees, poly-trees, directed acyclic graphs), efficient propagation algorithms have been proposed by Kim and Pearl [21]. However, the complexity of multiple-connected BBNs is still exponential in the number of nodes of the largest sub-graph. When a graph decomposition is not possible, we resort to approximate methods, such as clustering and bounding conditioning, and simulation techniques, such as logic samplings and Markov simulations.

Dempster-Shafer (DS) systems are a typical example of interval-valued probabilistic reasoning systems. They provide lower and upper probability bounds instead of a single value as in most BBN cases. The DS theory was developed independently by Dempster [22] and Shafer [23]. Dempster proposed a calculus for dealing with interval-valued probabilities induced by multiple-valued mappings. Shafer, on the other hand, started from an axiomatic approach and defined a calculus of belief functions. His purpose was to compute the credibility (degree of belief) of statements made by different sources, taking into account the sources' reliability. Although they started from different semantics, both calculi were identical.

Probabilistic computing provides a way to evaluate the outcome of systems affected by randomness (or other types of probabilistic uncertainty). PC's basic inferential mechanism - conditioning - allows us to modify previous estimates of the system's outcome based on new evidence.

**Comparison between Probabilistic and Fuzzy Computing.** In this brief review of fuzzy and probabilistic computing, we would like to emphasize that randomness and fuzziness capture two different types of uncertainty. In randomness, the uncertainty is derived from the non-deterministic membership of a point from a sample space (describing the set of possible values for the random variable), into a well-defined region of that space (describing the event). A probability value describes the tendency or frequency with which the random variable

takes values inside the region. In fuzziness, the uncertainty is derived from the deterministic but *partial* membership of a point (from a reference space) into an imprecisely defined region of that space. The region is represented by a fuzzy set. The characteristic function of the fuzzy set maps every point from such space into the real-valued interval [0,1], instead of the set {0,1}. A partial membership value does not represent a frequency. Rather, it describes the degree to which that particular element of the universe of discourse satisfies the property that characterizes the fuzzy set. In 1968, Zadeh noted the complementary nature of these two concepts, when he introduced the probability measure of a fuzzy event [24]. In 1981, Smets extended the theory of belief functions to fuzzy sets by defining the belief of a fuzzy event [25]. These are the first two cases of hybrid systems illustrated in Figure 1.

### Neural Computing

The genealogy of neural networks (NN) goes back to 1943, when McCulloch and Pitts showed that a network of binary decision units (BDNs) could implement any logical function [26]. Building upon this concept, Rosenblatt proposed a one-layer feedforward network, called a *perceptron*, and demonstrated that it could be trained to classify patterns [27-29]. Minsky and Papert [30] proved that single-layer perceptrons could only provide linear partitions of the decision space. As such they were not capable of separating nonlinear or non-convex regions. This caused the NN community to focus its efforts on the development of multilayer NNs that could overcome these limitations. The training of these networks, however, was still problematic. Finally, the introduction of backpropagation (BP), independently developed by Werbos [31], Parker [32], and LeCun [33], provided a sound theoretical way to train multi-layered, feed-forward networks with nonlinear activation functions. In 1989, Hornik et al. proved that a three-layer NN (with one input layer, one hidden layer of squashing units, and one output layer of linear units) was a universal functional approximator [34].

Topologically, NNs are divided into *feedforward* and *recurrent* networks. The feedforward networks include single- and multiple-layer *perceptrons*, as well as radial basis functions (RBF) networks [35]. The recurrent networks cover competitive networks, self-organizing maps (SOMs) [36], Hopfield nets [37], and adaptive resonance theory (ART) models [38]. While feed-forward NNs are used in supervised mode, recurrent NNs are typically geared toward unsupervised learning, associative memory, and self-organization. In the context of this paper, we will only consider feed-forward NNs. Given the functional equivalence already proven between RBF and fuzzy systems [39] we will further limit our discussion to multi-layer feed-forward networks. A comprehensive current review of neuro-computing can be found in [40].

Feedforward multilayer NNs are computational structures that can be trained to learn patterns from

examples. They are composed of a network of processing units or neurons. Each neuron performs a 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). By using a training set that samples the relation between inputs and outputs, and a learning method that trains their weight vector to minimize a quadratic error function, neural networks offer the capabilities of a supervised learning algorithm that performs fine-granule local optimization.

### Evolutionary Computing

Evolutionary computing (EC) algorithms exhibit an *adaptive* behavior that allows them to handle non-linear, high dimensional problems without requiring differentiability or explicit knowledge of the problem structure. As a result, these algorithms are very robust to time-varying behavior, even though they may exhibit low speed of convergence. EC covers many important families of stochastic algorithms, including *evolutionary strategies* (ES), proposed by Rechenberg [41] and Schwefel [42], *evolutionary programming* (EP), introduced by Fogel [43-44], and *genetic algorithms* (GAs), based on the work of Fraser [45], Bremermann [46], Reed et al. [47], and Holland [48-50], which contain as a subset *genetic programming* (GP), introduced by Koza [51].

The history of EC is too complex to be completely summarized in a few paragraphs. It could be traced back to Friedberg [52], who studied the evolution of a learning machine capable of computing a given input-output function; Fraser [45] and Bremermann [46], who investigated some concepts of genetic algorithms using a binary encoding of the genotype; Barricelli [53], who performed some numerical simulation of evolutionary processes; and Reed et al. [47], who explored similar concepts in a simplified poker game simulation. The interested reader is referred to [54] for a comprehensive overview of evolutionary computing and to [55] for an encyclopedic treatment of the same subject. A collection of selected papers illustrating the history of EC can be found in [56].

We will provide a brief description of four components of EC: ES, EP, GAs, GP. Evolutionary strategies were originally proposed for the optimization of continuous parameter spaces. In typical ES notation the symbols  $(\mu, \lambda)$ -ES and  $(\mu + \lambda)$ -ES indicate the algorithms in which a population of  $\mu$  parents generate  $\lambda$  offspring; the best  $\mu$  individuals are selected and kept in the next generation. In the case of  $(\mu, \lambda)$ -ES the parents are excluded from the selection and cannot be part of the next generation, while in the  $(\mu + \lambda)$ -ES they can. In their early versions, ES used a continuous representation and mutation operator working on a single individual to create a new individual, i.e.,  $(1+1)$ -ES. Later, a recombination operator was added to

the mutation operator as evolution strategies were extended to evolve a population of individuals, as in  $(\mu+1)$ -ES [57] and in  $(\mu+\lambda)$ -ES [58]. Each individual has the same opportunity to generate an offspring. Furthermore, each individual has its own mutation operator, which is inherited and may be different for each parameter.

In its early versions, evolutionary programming was applied to identification, sequence prediction, automatic control, pattern recognition, and optimal gaming strategies [43]. To achieve these goals, EP evolved finite state machines that tried to maximize a payoff function. However, recent versions of EP have focused on continuous parameter optimization and the training of neural networks [59]. As a result, EP has become more similar to ES. The main distinction between evolutionary programming and evolutionary strategies is that EP focuses on the behavior of species, while ES focus on the behavior of individuals. Therefore, EP does not usually employ crossover as a form of variation operator. EP can be considered as a special case of  $(\mu+\mu)$ -ES, in which mutation is the only operator used to search for new solutions.

Genetic algorithms perform a randomized global search in a solution space by using a genotypic rather than phenotypic representation. Canonical GAs encode each candidate solution to a given problem as a binary, integer, or real-valued string, referred to as chromosome. Each string's element represents a particular feature in the solution. The string is evaluated by a fitness function to determine the solution's quality: better-fit solutions survive and produce offspring, while less-fit solutions are culled from the population. To evolve current solutions, each string can be modified by two operators: crossover and mutation. Crossover acts to capture the best features of two parents and pass it to an offspring. Mutation is a probabilistic operator that tries to introduce needed features in populations of solutions that lack such features. The new individuals created by these operators form the next generation of potential solutions. The problem constraints are typically modeled by penalties in the fitness function or encoded directly in the solution data structures [60].

Genetic programming is a particular form of GA, in which the chromosome has a hierarchical rather than a linear structure whose size is not predefined. The individuals in the population are tree structured programs, and the genetic operators are applied to their branches (subtrees) or to single nodes. This type of chromosome can represent the parse tree of an executable expression, such as an S-expression in LISP, an arithmetic expression, etc. GP has been used in time-series predictions, control, optimization of neural network topologies, etc.

As noted by Fogel [54], ES, EP, and GAs share many common traits: "...Each maintains a population of trial solutions, imposes random changes to those solutions, and incorporates 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.”

Finally, we would like to remark that EC components have increasingly shared their typical traits: ES have added recombination operators similar to GAs, while GAs have been extended by the use of real-number-encoded chromosomes, adaptive mutation rates, and additive mutation operators. For the sake of brevity, in the context of this paper, we will limit most of our discussion to genetic algorithms, but encourage the reader to consider evolutionary algorithms broadly.

### Soft Computing Taxonomy

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, such as theorem provers, can be used to attach binary truth-values to statements that describe the state or behavior of the physical system.

When we solve real-world problems, we realize that such systems are typically ill-defined, difficult to model, and possess large solution spaces. In these cases, precise models are impractical, too expensive, or non-existent. Our solution must be generated by leveraging two kinds of resources: *problem domain knowledge* of the process or product and *field data* that characterize the behavior of the system. The relevant available domain knowledge is typically a combination of first principles and empirical knowledge, and is usually incomplete and sometimes erroneous. The available data are typically a collection of input-output measurements, representing instances of the system's behavior, and may be incomplete and noisy.

We can observe from Figure 1 that the two main approaches in soft computing are *knowledge-driven* reasoning systems (such as probabilistic and fuzzy computing) and *data-driven* search and optimization approaches (such as neuro and evolutionary computing). This taxonomy, however, is soft in nature, given the existence of many hybrid systems that span across more than one field.

### 1.4 Alternative Approaches to SC

The alternative approaches to SC are the traditional knowledge-driven reasoning systems and the data-driven systems. The first class of approaches are exemplified by first-principle-derived models (based on differential or difference equations), by first-principle-qualitative models (based on symbolic, qualitative calculi [61-62], by classical

Boolean systems, such as theorem provers (based on unification and resolution mechanisms), or by expert systems embodying empirical or experiential knowledge. All these approaches are characterized by the encoding of problem domain knowledge into a model that tries to replicate the system's behavior. The second class of approaches are the regression models and crisp clustering techniques that attempt to derive models from any information available from (or usually buried in) the data.

Knowledge-driven systems, however, have limitations, as their underlying knowledge is usually incomplete. Sometimes, these systems require the use of simplifying assumptions to keep the problem tractable (e.g., linearization, hierarchy of local models, use of default values). Theoretically derived knowledge may even be inconsistent with the real system's behavior. Experiential knowledge, on the other hand, could be static, represented by a collection of instances of relationships among the system variables (sometimes pointing to causality, more often just highlighting correlation). The result is the creation of precise but simplified models that do not properly reflect reality, or the creation of approximate models that tend to become stale with time and are difficult to maintain.

Purely data-driven methods also have their drawbacks, since data tend to be high-dimensional, noisy, incomplete (e.g., databases with empty fields in their records), or wrong (e.g., outliers due to malfunctioning or failing sensors, transmission problems, erroneous manual data entries). Some techniques have been developed to address these problems, such as feature extraction, filtering and validation gates, imputation models, and virtual sensors that model the recorded data as a function of other variables.

The fundamental problem of these classical approaches lies in representing and integrating uncertain, imprecise knowledge in data-driven methods or in making use of somewhat unreliable data in a knowledge-driven approach.

### 1.5 SC Solutions

Although it would be presumptuous to claim that soft computing *solves* this problem, it is reasonable to affirm that SC provides a different paradigm in terms of representation and methodologies, which facilitates these integration attempts. For instance, in classical control theory the problem of developing models is decomposed into system identification and parameter estimation. Usually the former is used to determine the order of the differential equations and the latter determines its coefficients. Hence, in this traditional approach we have ***model = structure + parameters***. This equation does not change with the advent of soft computing. However, we now have a much richer repertoire to represent the structure, to tune the parameters, and to iterate this process. It is understood that the search method used to find the parameter values is an important and implicit part of the

above equation, which needs to be chosen carefully for efficient model construction.

For example, the knowledge base (KB) in a Mamdani-type fuzzy system [63] is typically used to approximate a relationship between a state  $X$  and an output  $Y$ . The KB is completely defined by a set of *scaling factors* (SF), determining the ranges of values for the state and output variables; a *termset* (TS), defining the membership function of the values taken by each state and output variable; and by a *ruleset* (RS), characterizing a syntactic mapping of symbols from  $X$  to  $Y$ . The *structure* of the underlying model is the ruleset, while the model *parameters* are the scaling factors and termsets. The inference obtained from such a system is the result of interpolating among the outputs of all relevant rules. The inference's outcome is a membership function defined on the output space, which is then aggregated (defuzzified) to produce a crisp output. With this inference mechanism we can define a deterministic mapping between each point in the state space and its corresponding output. Therefore, we can now equate a fuzzy KB to a response surface in the cross product of state and output spaces, which approximates the original relationship.

A Takagi-Sugeno-Kang (TSK) type of fuzzy system [64] increases its representational power by allowing the use of a first-order polynomial, defined on the state space, to be the output of each rule in the ruleset. This enhanced representational power, at the expense of local legibility [65], results in a model that is equivalent to radial basis functions [66]. The same model can be translated into a structured network, such as the adaptive neural fuzzy inference systems (ANFIS) proposed by Jang [67]. In ANFIS the ruleset determines the topology of the net (model structure), while dedicated nodes in the corresponding layers of the net (model parameters) define the termsets and the polynomial coefficients. Similarly, in the traditional neural networks the topology represents the model structure and the links' weights represent the model parameters.

While NN and structured nets use local search methods, such as backpropagation, to tune their parameters, it is possible to use evolutionary computation based global search methods to achieve the same parametric tuning or to postulate new structures. An extensive coverage of these approaches can be found in [8].

## 1.6 Paper Structure

In the next section we will focus on SC hybrid algorithms and the synergy derived from their use of complementary components. After a brief discussion of the problem of diagnosis, estimation, and control provided in Section 3, we will illustrate three applications of SC techniques. The first application, described in Section 4, consists in the adaptation of fuzzy clustering to classify gas turbine anomalies. The second application, illustrated in Section 5, covers the use of genetic algorithms to tune a fuzzy system

that implements an automatic train handler controller. The third application, discussed in Section 6 and 7, describes the use of a customized neural-fuzzy system and a fusion of models to determine the market value of residential properties. In the last section we summarize the advantages of using soft computing hybrid systems and discuss some potential extensions of these technologies.

## 2. Hybrid Algorithms: Symbiosis

Over the past few years we have seen an increasing number of hybrid algorithms, in which two or more soft computing technologies have been integrated to leverage the advantages of individual approaches. By combining smoothness and embedded empirical qualitative knowledge with adaptability and general learning ability, these hybrid systems improve the overall algorithm performance. We will now analyze a few of such combinations, as depicted in the lower box of Figure 1. First we will illustrate the application of NNs and EC to tune FL systems, as well as the use of EC to generate and tune NNs. Then, we will see how fuzzy systems can control the learning of NNs and the run-time performance of EC.

### 2.1 FL tuned by NN

ANFIS [67] is a representative hybrid system in which NNs are used to tune a fuzzy logic controller (FLC). ANFIS consists of a six-layer 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 will add up to one. The fifth layer computes the polynomial coefficients in the right-hand-side of each Takagi-Sugeno rule. 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 that is compared with the one from the training set to produce an 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.

### 2.2 FL Tuned by EC

As part of the design of a fuzzy controller, it is relatively easy to specify a cost function to evaluate the state trajectory of the closed-loop system. On the other hand, for each state vector instance, it is difficult to specify its corresponding desired controller's actions, as would be required by supervised learning methods. Thus evolutionary algorithms can evolve a fuzzy controller by using a fitness function based on such a cost function.

Many researchers have explored the use of EC to tune fuzzy logic controllers. Cordon et al. [68] alone list a 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 the sake of brevity we will limit this section to a few contributions. These methods differ mostly in the order or the selection of the various FLC components that are tuned (termsets, rules, scaling factors).

Karr, one of the precursors in this quest, used GAs to modify the membership functions in the termsets of the variables used by the FLC [69]. 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.

Lee and Takagi tuned both rules and termsets [70]. They used a binary encoding for each three-tuple characterizing a triangular membership distribution. Each chromosome represents a TSK-rule, concatenating the membership distributions in the rule antecedent with the polynomial coefficients of the consequent.

Most of the above approaches used the sum of quadratic errors as a fitness function. Surmann et al. extended this quadratic function by adding to it an entropy term describing the number of activated rules [71].

Kinzel et al. departed from the string representation and used a cross-product matrix to encode the rule set (as if it were in table form) [72]. They also proposed customized (point-radius) crossover operators that were similar to the two-point crossover for string encoding. They first initialized the rule base according to intuitive heuristics, used GAs to generate a 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 [73].

Bonissone et al. [74] followed the tuning order suggested by Zheng [75] for manual tuning. They began with macroscopic effects by tuning the FLC state and control variable *scaling factors* while using a standard uniformly spread termset and a homogeneous rule base. After obtaining the best scaling factors, they proceeded to tune the termsets, causing medium-size effects. Finally, if additional improvements were needed, they tuned the *rule base* to achieve microscopic effects. 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.

This approach exemplifies the synergy of SC technologies, as it was used in the development of a fuzzy PI-control to synthesize a freight train handling controller. This application is described in greater detail in Section 5.

### 2.3 NNs Generated by EC

Evolutionary computing has been successfully applied to the synthesis and tuning of neural networks. Specifically, EC has been used to:

1. Find the optimal set of weights for a given topology, thus replacing backpropagation;
2. Evolve the network topology (number of hidden layers, hidden nodes, and number of links), while using backpropagation to tune the weights;
3. Simultaneously evolve both weights and topology;
4. Evolve the reward function, making it adaptive;
5. Select the critical input features for the neural network.

An extensive review of these five application areas can be found in [76-78]. Here, we will briefly cover three early applications of EC to NNs, focusing on the evolution of the network's weights and topology. Backpropagation (BP), the typical local tuning algorithm for NNs, is a gradient-based method that minimizes the sum of squared errors between ideal and computed outputs. BP has an efficient convergence rate on convex error surfaces. However, when the error surface exhibits plateaus, spikes, saddle-points, and other multimodal features, BP, like other gradient-based methods, can become trapped in sub-optimal regions. Instead of repeating the local search from many different initial conditions, or applying perturbations on the weights when the search seems to stagnate, it is desirable to rely on a more robust, global search method. In 1988, Jurick provided a critique of backpropagation and was among the firsts to suggest the use of robust tuning techniques for NNs, including evolutionary approaches [79].

**Using GAs.** Montana and Davis proposed the use of genetic algorithms to train a feedforward NN for a given topology [80]. They used real-valued steady state GAs with direct encoding and additive mutation. GAs perform efficient coarse-granularity search (finding the promising region where the global minimum is located) but they may be inefficient in the fine-granularity search (finding the global minimum). These characteristics motivated Kitano to propose a hybrid algorithm in which the GAs would find a *good* parameter region that was used to initialize the NN [81]. At that point, BP would perform the final parameter tuning. McInerney improved Kitano's algorithm by using the GA to escape from the local minima found by the BP during the training of the NNs (rather than initializing the NNs using the GAs and then tuning it using BP) [82]. They also provided a dynamic adaptation of the NN learning rate.

The simultaneous evolution of the network's weights and topology represents a more challenging task for the GAs. Critical to this task's success is the appropriate selection of the genotypic representation. There are three available methods: *direct*, *parametrized*, and *grammar* encoding.

*Direct binary encoding* is often implemented by encoding the network as a connectivity matrix of size  $N \times N$ ,

where  $N$  is the maximum number of neurons in the network. The encoding of the weights requires predetermining the granularity needed to represent the weights. Typically, direct binary encoding results in long chromosomes that tend to decrease the GA's efficiency. To address this problem, Maniezzo [83] suggested the use of variable granularity to represent the weights. Specifically, he proposed using different-length genotypes that included a control parameter, the coding granularity. He also recognized the disruptive effect of the crossover operation and decreased the probability of crossover, relying more on the mutation operator for the generation of new individuals.

*Parametrized encoding* uses a compact representation for the network, indicating the number of layers and the number of neurons in each layer, instead of the connectivity matrix [84-85]. However, this type of encoding can only represent layered architectures, e.g., fully connected networks without any connection from input to output layers.

*Grammar encoding* is the genotypic representation that seems to have the best scalability properties among GAs. There are two types of grammar encoding: *matrix grammar*, and *graph grammar*. Kitano proposed the matrix grammar encoding, in which the chromosomes encode a graph generation grammar [86]. A grammar rule rewrites a  $1 \times 1$  matrix into a  $2 \times 2$  matrix, and so on, until it generates a  $2^k \times 2^k$  matrix, such that  $2^k < N$ , the maximum number of units in the network. An  $N \times N$  matrix is extracted from the  $2^k \times 2^k$  matrix. Each character is mapped into 0 or 1, to generate the network's connectivity matrix. Unfortunately, this encoding is not very efficient since only a small percentage of the rules are used. Furthermore, in the case of feed-forward NNs, less than half of the connectivity matrix is used. Recently, Siddiqi and Lucas [88] showed that grammar encoding is superior to direct encoding when the initial population of network topologies is formed with sparsely connected individuals (with a probability of connection = 0.3). However, the results produced by direct encoding for densely connected networks (with a probability of connection = 0.7) were indistinguishable from Kitano's grammar encoding.

Grau proposed a graph grammar called *cellular encoding* [88]. He used an attribute grammar in which the grammar rules are represented by a tree structure. Each rewrite rule provides an operator that computes the attributes of a symbol on the rule's right-hand side from the attributes of the symbols on the rule's left-hand side. This approach was used to generate large size Boolean networks.

**Using GP.** Koza and Rice proposed another solution to some of GA's perceived problems [89]. They advocated the use of genetic programming, which does not rely on a predefined chromosome's length. GP uses a direct encoding of the NN's weights and topology in a genetic tree structure.

**Using EP.** The ability of determining the model's structure using evolutionary algorithms was proposed by

Fogel [90] and McDonnell and Waagen [91], among others. The latter suggested using EP with a fitness function that combines a measure of the network's complexity (number of connections) with the mean sum-squared pattern error. This fitness favored removing those synapses that had small effect on the pattern error. Angeline et al. [59] argued that GAs were inefficient in evolving network topologies due to the deception problem [92-93] and the complexity of the interpretation function. Angeline proposed an EP, called GeNeralized Acquisition of Recurrent Links (GNARL). This system relies on a number of mutation operators that allow for the simultaneous structural and parametric optimization of the network. This evolutionary program was capable to generate networks with complex, non-symmetrical topologies. Their view has been followed by many other researchers, such as Lee and Kim [94], who proposed the use of EP with a linked-list encoding to avoid the permutation problem caused by the many-to-one mapping between genotypes and phenotypes in GAs.

## 2.4 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. This characteristic has been the basis for the successful development and deployment of fuzzy controllers.

Typically this knowledge is used to synthesize fuzzy controllers for dynamic systems [1]. However, in this case the knowledge is used to implement a smart algorithm-controller that allocates the algorithm's resources to improve its convergence and performance. As a result, fuzzy rule bases and fuzzy algorithms have been used to monitor the performance of NNs or EC 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.

The learning rate is a function of the step size and determines how fast the algorithm will move along the error surface, following its gradient. Therefore the choice of the learning rate has an impact on the accuracy of the final approximation and on the speed of convergence. The smaller its value the better the approximation but the slower the convergence. The momentum represents the fraction of the previous changes to the weight vector, which will still be used to compute the current change. As implied by its name, the momentum tends to maintain changes moving along the same direction thus preventing oscillations in shallow regions of the error surface and often resulting in faster convergence. Jacobs established a heuristic rule, known as the *delta-bar-delta* rule to increase the size of the learning rate if the sign of the error gradient was the same over several consecutive steps [95]. Arabshahi et al. developed a simple fuzzy controller to

modify the learning rate as a function of the error and its derivative, considerably improving Jacobs' heuristics [96].

The selection of these parameters involves a tradeoff: in general, large values of learning rate and momentum result in fast error convergence, but poor accuracy. On the contrary, small values lead to better accuracy but slow training, as proved by Wasserman [97].

## 2.5 EC Controlled by FL

The use of fuzzy logic to translate and improve heuristic rules has also been applied to manage EC resources (population size, selection pressure, probabilities of crossover and mutation), during their transition from *exploration* (global search in the solution space) to *exploitation* (localized search) in the discovered regions of that space that appear to be promising [68,70].

The management of EC resources gives the algorithm an adaptability that improves its efficiency and convergence speed. Herrera and Lozano suggest this adaptability can be used in the GA's parameter settings, the selection of genetic operators, solution representation, and fitness function [98].

The crucial aspect of this approach is to find the correct balance between the computational resources 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 generic enough to be applicable to other object-level problem domains and if its run-time overhead is offset by the run-time performance improvement of the algorithm. An example of an application of this type of hybrid system can be found in [99].

## 2.6 Advantages of SC Hybrid Systems

This brief review of hybrid systems illustrates the interaction of knowledge and data in SC. To tune *knowledge-derived models* we first translate domain knowledge into an initial structure and parameters and then use global or local data search to tune the parameters. To control or limit search by using prior knowledge we first use global or local search to derive the models (structure + parameters), we embed knowledge in operators to improve global search, and we translate domain knowledge into a controller to manage the solution convergence and quality of the search algorithm. All these facets have been exploited in some of the service and control applications described in the following sections.

## 3. SC Applications for Predictive Modeling: Diagnostics, Control, and Estimation

The task of predictive modeling is to forecast a course of events from a set of observations. For instance, predicting aircraft engine performance from exhaust gas temperatures (EGT) and fan speeds, or predicting the value of a residential property from the age and location of the house.

In essence, predictive modeling synthesizes the nonlinear mapping from the inputs to the outputs. In the above examples, the input is the EGT and fan speed of an engine, or the age and location of a house, while the output is the engine performance or the value of the property, respectively.

Predictive modeling is the focus of many industrial and commercial applications. It can be used for *diagnosis* in which a set of fault hypotheses is identified - e.g., EGT exceeds its margin based on aircraft engine performance - and the system solves a classification problem (as in the first application that we discuss in Section 4). It can also be used for control in which the system has to output control actions that will impact the future state of the system (as in our second example, involving freight train control in Section 5). Predictive modeling can also be used for *estimation* of the market value of a residential property, based on the attributes of the house (as in the last application, in Sections 6 and 7).

## 4. Classifying Gas Turbine Anomalies with Adaptive Fuzzy Clustering

### 4.1 Problem Description

Gas turbines are used in applications such as stationary power plants, airplanes, and helicopters. Depending on the particular use, the design and size of the gas turbine will vary. On a coarse level, however, gas turbines use the same operating principles. It is desirable to be able to detect incipient failures before they cause costly secondary damage or result in equally undesired shutdowns.

Service providers have long tracked the behavior of engines by measuring many system parameters and by using trend analysis to detect changes that might be indicative of developing failures [100]. Challenges of these schemes are that faults may not be recognized distinctively due to large amounts of noise as well as changing operating conditions that constantly move the estimate for "normal" operation. The former are caused in part by changing environmental conditions for which corrections with first principle models or regression models work only to some extent. The latter are due in part to changes of schedules, maintenance, etc., which are not necessarily known to the analyst.

In trend analysis, online sensor data and parameter settings are evaluated for deviations from normal operating conditions. When threshold settings are tripped and if the reading is not considered an outlier, an alarm is raised. The settings used in trend analysis are typically below the limit that would cause an immediate shutdown and are meant to create awareness of potential problems before they turn into events that result in revenue loss (for airplanes, that could mean grounding of an airplane, damage to the engine, departure delays, etc.) An alarm normally causes the analyst to examine the trend charts to see if it appears

that something real has happened. If something does appear to be suspicious, remedial actions (borescoping, engine washing, overhauls, etc.) are suggested. Problems with the system include the improper identification of alarms with engines where data quality is “bad” (data quality varies considerably between different engines due to different operating conditions and data acquisition) and where data drift due to wear. In addition, noise from poorly calibrated and deteriorating sensors and faulty data acquisition, among others, can result in excess generation of some alarms. If too many alarms are generated, the user must look at trends more often than desired to weed out the faulty alarms from the true ones. Increasing the thresholds will typically also increase the false negatives (missing an alarm condition) rate, which is typically even less desirable than a large number of false positives.

## 4.2 Approach

The approach for adaptive clustering combines multivariate data evaluation using fuzzy clusters with an exponential filter, which lets the clusters adapt to changing environments. The adaptive properties of the clusters allow the centroids to move and their shape to vary.

## 4.3 Prior Work

Karamouzis and Feyock proposed a system integrating model based and case based reasoning techniques [101]. This system contained a self-organizing memory structured as a frame-based abstraction hierarchy for storing previously encountered problems. The cases were aircraft accident reports. Sensor inputs were gas temperature, engine pressure ratio, and human input such as sound of explosion and smell of smoke in passenger cabin. The time sequence was also of importance in establishing a causal relation such as an aberrant reading of the fan speed before an abnormal observation of the compressor speed. The system incorporated a functional dependency implemented via a digraph and a causality submodel describing transitions between various states of the system. The system searched its case library for the most similar case by a weighted count of corresponding symptoms and gave higher degree of similarity for symptoms occurring early in the fault occurrence.

Reibling and Bublin proposed a system that integrates pattern-matching techniques with causal networks for OMS to model normal and abnormal behavior [102].

Fernandez-Montesinos used Kohonen’s feature maps and expert systems to support the interpretation of in-flight monitoring data and to ensure more consistent engine condition monitoring (ECM) [103]. ECM was supported by performance trend analysis software (ADEPT) which captured deterioration and failure of engine parts. Shift patterns allowed the localization of problems within the engine modules. The two main tasks tackled were recognition of a pattern indicating a possible problem, and the interpretation and further analysis. Kohonen’s feature maps were chosen because of their self-organizing

properties and the incremental extension of the domain of the patterns. Output of the Kohonen’s feature map was an error code that indicated whether the module recognized a problem. As starting points for the network, fingerprints provided by the engine manufacturer (GE) were used. An expert system controlled the proposed neural net and interpreted the output.

Records and Choi [104] investigated strategies to filter spurious symptoms in aircraft engine fault monitoring systems (resulting from the monitoring system’s inability to accurately assess the expected value from an engine model). They used a back-end knowledge based approach and a front-end neural network that generated expectation values for the monitored sensor.

Gomm [105] and Patel [106] both suggested the use of a self-adaptive neural network with on-line learning capabilities. Gomm used radial basis function networks to which new output nodes were automatically added when a new process fault is encountered. Adaptation was achieved using recursive linear algorithms that train the localized network parameters.

## 4.4 Solution Description

### 4.4.1 Fuzzy Clusters

Evaluation in multivariate feature space helps in distinguishing some faults because system variables are interconnected and changes in one variable may result in changes of other variables as well. For example, engine variables such as exhaust gas temperature ( $EGT$ ), fuel flow ( $w_f$ ), turbine speed ( $n_2$ ) are correlated such that particular failure conditions show up in changes of several variables. A bleed problem, where air leaks from the compressor, will result in a less efficient engine. However, in case of an aircraft engine, the controller demands a certain thrust level and will cause more fuel to be injected into the combustors. This in turn will raise the turbine speed as well as the exhaust gas temperature. The change in any of the engine parameters may be too small to be recognized alone. However, the change is more pronounced in a higher-dimensional space, and should therefore be detected more easily. Still, depending on the amount of noise, an overlap remains in between the cluster centers where variables cannot be partitioned clearly and the potential for misclassification remains high, making a crisp classifier not very feasible. The answer is not necessarily adding more features to the classification scheme because the decreasing potential information gain (in Euclidean space) is blurred additionally by decreasing feature quality, assuming one starts out using the highest-quality feature and adding others in decreasing order of value. At a certain threshold, more features may actually decrease the performance.

Engine behavior is clustered into different regions in the  $EGT-w_f-n_2$  space. Data behaving normally will appear in a cluster labeled “normal.” The presence of an alarm condition, such as a rapid change of  $EGT$  caused by a compressor leak is located in the leak fault cluster. Rate of

change of the data is not included here using the assumption that the abnormal condition will show up at least an order of magnitude faster than a slow drift (which is considered normal). The clusters are of non-uniform and nonlinear degrading size and shape. This allows data of a training set with a nonlinear boundary between classes to be separated more easily. Moreover, the boundaries between the clusters are not crisp. That is, the evaluation of the cluster membership is carried out such that the degree of membership is largest at the cluster center, with typically (but not necessarily) nonlinear slope outward. We use a fuzzy *c*-means clustering algorithm [107] for this purpose.

#### 4.4.2 Adaptation

Normal behavior is constantly redefined. Thermal, chemical, and mechanical wear degrade the performance of the engine. These changes are expected but not necessarily predictable because they are driven partly by external factors. In addition, maintenance, controller software changes, and other factors may vary the operating conditions as well. At any rate, these changes must also be reflected in changes of the clusters to retain desirable classification properties. To achieve this goal, the centroid was superimposed with an exponential filter with small smoothing parameter, which forces the centroid to be updated according to the current data. Data were used for updating only when they were classified as part of a particular cluster. That is, data of type “normal” were used to update the cluster “normal” but not the cluster “leak fault” and vice versa. After updating of the memberships in the cluster (which calculates membership values and cluster centers), the adaptation of a fuzzy cluster is performed by borrowing the exponential weighted moving average filter from statistical signal processing, which we express as:

$${}^{new}\tilde{v} = {}^{old}\tilde{v} + \tilde{\alpha} \left( \frac{\sum_{i=1}^n \tilde{u}_i^m y_i}{\sum_{i=1}^n \tilde{u}_i^m} - {}^{old}\tilde{v} \right)$$

where

superscript “ $\sim$ ” denotes the winner

superscript “ $\hat{\sim}$ ” denotes the loser

${}^{new}\tilde{v}$  is the new cluster position

${}^{old}\tilde{v}$  is the old cluster position

$\tilde{\alpha}$  is the adaptive learning coefficient calculated (for two clusters) by

$$\tilde{\alpha} = \alpha(1 - |\tilde{u} - \hat{u}|)$$

where

$\alpha$  is a constant term, e.g.,  $\alpha=0.05$

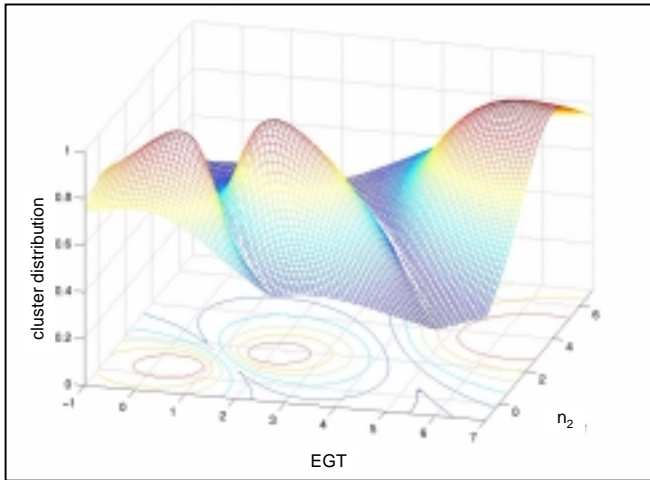
$u$  is the membership value

$y_i$  is the data point under consideration

The adaptation rate, given by  $\tilde{\alpha}$ , tries to minimize ambiguity. When the degree of membership in a winning cluster is medium (ambiguous),  $\tilde{u}$  and  $\hat{u}$  are almost the same. In that case,  $\tilde{\alpha}$  is large, thus trying to untangle the indeterminate situation, and moving the winning cluster farther toward the new data point. Moreover, it causes a higher rate of change near the true fault cluster boundaries and less change where the fault classification is already working well. Thus, it has the effect of focusing its learning where it is needed most, as dictated by recent data. As a result, the centroids move with changing normal conditions. The scheme will dampen the effects of noise as well, since it gives some weight to past data and acts as an averaging scheme. It will also, like all filters, lag behind true changes in the system.

#### 4.4.3 Alertness Filter

Another approach to increase robustness is to track the cumulative occurrence of “suspicious” readings, rather than tripping the alarm after just one “fault” classification. We propose an *alertness filter* for doing this, which forces the reoccurrence of several suspicious readings before an alarm is issued. If only one suspicious reading is encountered, the value of an “alertness counter” - which would be zero under normal conditions - is incremented. If a second suspicious reading is encountered immediately afterwards, the counter value goes up. Otherwise, it goes down. In the latter case, the first suspicious reading is treated just as an outlier and no maintenance is recommended. If it is not an outlier, i.e., the condition prevails, the level of the alertness will go up until a full alarm is issued, at which point remedial action will be advocated. At the alarm threshold, further fault readings will not increase the alertness level. Instead, it will be limited to a user-specified number. This saturation avoids lengthy recovery from alarm conditions when the system recovers either spontaneously or through maintenance. The purpose of the alertness filter is to further improve the classification rate of the system.

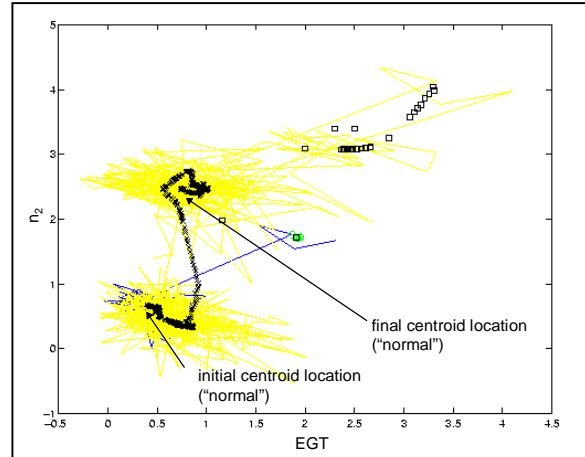


**Figure 2: Fuzzy clusters for several conditions displayed in  $EGT-n_2$  space**

#### 4.5 Results

The classification system was tested with historical gas turbine data. Different events of similar fault conditions were examined and divided into training and test cases. For training, the use of both deliberate placement and random seed points (for the cluster centroids) was investigated. Placing the starting points into the general expected neighborhood resulted in faster learning and convergence. Convergence was on average accomplished after about 20 iterations, depending on the start conditions. To prevent a fitting bias for normal conditions due to a vastly larger amount of data for that state, those data were undersampled so that “normal” and “faulty” operation were equally represented. Figure 2 shows the distribution of clusters after training in  $EGT-n_2$  space.

After training, the system was used for classification of on-line test data. The false negative rate is as low as with traditional trending tools (none observed with the test data available) and the false positive rate improved from 95% to less than 1%. Figure 3 shows the adaptation of a cluster during operation.



**Figure 3: Adaptation of clustering during operation**

The “x” denotes the location of the centroid of cluster “normal” and the squares denote the location of a fault cluster. During a software change, the normal region changes considerably in one dimension as seen by the vertical jump of the operating point upward in the graph. Because there were no changes in the other two dimensions, the algorithm adapts correctly to the new location while retaining the ability to detect faults. The drastic change also shows the lag introduced by the exponential filter.

#### 4.6 Summarizing the SC Application to Diagnostics

We have presented the use of adaptive fuzzy clusters for fault classification of gas turbine engine sensor data. The advantages of this system are its ability to deal with extremely noisy sensors and to adjust to unpredictable slow drift. Dramatic improvement in performance was demonstrated during testing. Future work could focus on detecting new fault types and the need for adaptively adding clusters.

In the broader context of soft computing and the theme of model = *structure* + *parameters*, the model structure here is provided by the fuzzy clusters (their number, which is a structural feature of the model). The model parameters are the centroid locations and membership functions, which are determined and tuned by a simplified Kalman filter type approach (exponential smoothing). In general, this type of search method is not provably optimal, unless certain assumptions about the system noise and behavior are met. The hybridization is a complementary one, in that two different technologies are achieving two functionally different aims.

Of course, it is possible to use other soft computing methods for this type of problem. For instance, EC could be used to determine the optimal centroid cluster locations in an offline computation using a limited set of initial data, but ongoing, real-time tuning of the clusters is currently impractical (if not unfeasible) if done by evolutionary

algorithms. The clustering could be implemented as a radial basis function neural network with competitive learning schemes (such as Learning Vector Quantization and its variants). Probabilistic schemes (with subjective probabilities) could also be employed here for ranking cluster candidates, and indeed BBNs have been used widely in diagnostic applications in industry. The fuzziness used here lends itself to human interpretation of the clusters in some cases, which is an added benefit.

## 5. Automated Train Handling Using Genetically Tuned Fuzzy Systems

### 5.1 Problem Description

The development of an automated train handler requires the control of a massive, distributed system with little sensor information. Freight trains consist of several hundred heavy railcars connected by couplers. A typical freight train can be as long as two miles and requires up to four coupled locomotives (12,000 hp) to pull it. Each coupler between railcars may have a dead zone and a hydraulically damped spring. This implies that the railcars can move relative to each other while in motion, leading to a train that can change its length by 50 – 100 feet. The controlled forces on the train are due to a throttle and dynamic brake exerting a force on the locomotive, which is transmitted through the couplers, and a distributed air brake. Handling of these controls has a direct effect on the intercar coupler dynamics and the forces and distances therein (called slack). The position of the cars and couplers cannot be electronically sensed. Couplers are subjected to high static forces and dynamic forces, which may lead to breakage of the coupler, the brake pipe, and the train. Violation of speed limits and excessive acceleration/braking may lead to derailment and severe cargo damage. Smooth handling while following a speed target is therefore absolutely imperative. These boundary conditions make it hard to control the train and the current completely manual control depends heavily on the experience of the crew.

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.

An automated system has to satisfy multiple goals: providing a certain degree of train handling uniformity across all crews, enforcing railroad safety rules (such as posted speed limits), maintaining the train schedule within small tolerances, operating the train in fuel-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. As described above, the handling of freight trains involves a multibody problem and proper slack management, without sensors for most of the state. These constraints lead to a complex problem that cannot be solved by the simpler schemes used by cruise controllers for other vehicles, such as cars, trucks, boats, etc.

### 5.2 Approach

Our design uses a fuzzy controller, composed of a cruise planning module and a cruise control module that can automate the controls of a freight train [74]. This system would be applicable during most of the train journey, except for the initial and final transients, i.e., the train starting and stopping. The planning module focuses on creating a good reference trajectory that the fuzzy controller tries to track. Here, we focus only on the use of 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. 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.

### 5.3 Prior Work

There is no significant prior work on automatic train handling for the type of general terrain use that we are targeting here. Our proposed solution involves a hybridization of fuzzy control and genetic algorithms. Please refer to Section 2.2 for a review of the prior work in GA based tuning of Fuzzy Controllers.

### 5.4 Solution Description

#### 5.4.1 Architecture

The overall scheme for the proposed train handling is shown in Figure 4. It consists of a fuzzy proportional integral controller (FPI) closing the loop around the train simulation (TSIM), using only the current velocity as its state input. This velocity and look-ahead 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 choices for the FPI parameters. The simulator TSIM is an in-house implementation, combining internal data with physical and empirical models.

The FPI controller uses the tracking error ( $e$ ) and its error change ( $\Delta e$ ) to recommend a change in the control outputs throttle notch and brake settings ( $\Delta u$ ). If allowed, the FPI will track the reference profile accurately. The computation of the error used by the FPI incorporates a look-ahead 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 velocity is known from the profile. Finally, the PI is tuned (off-line) using GAs that modify the controller's most sensitive parameters: scaling factors (SF) and membership function parameters (MF).

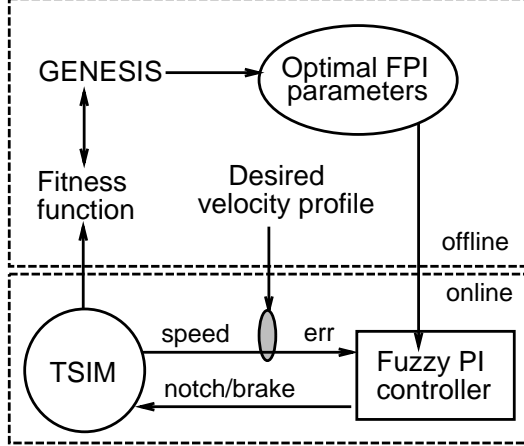


Figure 4: System schematic for using a GA to tune a fuzzy train controller.

#### 5.4.2 Designing the Fuzzy PI Controller

To summarize the controller briefly, it is described by a knowledge base (KB) composed of a rule set (RS), termsets of membership functions (MF), and scaling factors (SF). The rule set maps linguistic descriptions of state vectors  $[e; \Delta e]$  into incremental control actions  $\Delta u$ ; 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. The relationship between output variable,  $u$ , and control error,  $e$ , can be expressed approximately as follows [75]:

$$\begin{aligned} \frac{\Delta u(t)}{S_u} &\approx \frac{\Delta e(t)}{S_d} + \frac{e(t)}{S_e} \\ u(t) &\approx \frac{S_u}{S_d} e(t) + \frac{S_u}{S_e} \int e(t) dt \\ -S_e &\leq e(t) \leq S_e \\ -S_d &\leq \Delta e(t) \leq S_d \\ -S_u &\leq \Delta u(t) \leq S_u \end{aligned}$$

where  $S_e$ ,  $S_d$  and  $S_u$  are the scaling factors of error, change of error, and incremental output variable, respectively. On the other hand, a conventional PI controller has  $u(t) = K_p e(t) + K_i \int e(t) dt$  where  $K_p$  and  $K_i$  are the proportional and integral gain factors, respectively. Comparing the above equations, we obtain:

$$K_p \approx S_u / S_d, \quad K_i \approx S_u / S_e$$

#### 5.4.3 Tuning the Fuzzy PI

For the purposes of this study, GENESIS (GENETic Search Implementation System) has been adopted as a software development tool [108]. The user needs to provide only a *fitness function* that returns a corresponding value for any point in the search space.

**Fitness Functions.** To study the effects of different objectives, we investigated two fitness functions:

$$\begin{aligned} f_1 &= - \left( \sum_i |notch_i - notch_{i-1}| + |brake_i - brake_{i-1}| \right) \\ f_2 &= - \left( w_1 \frac{\sum_i |notch_i - notch_{i-1}|}{K_1} + w_2 \frac{\sum_i |v_i - v_i^d|}{K_2} \right) \end{aligned}$$

where  $v^d$  denotes the desired reference velocity and  $i$  is a distance or milepost index. The function  $f_1$  captures throttle jockeying of the throttle and brakes. The function  $f_2$  combines a weighted sum of velocity profile tracking accuracy and throttle jockeying. The train simulator TSIM provided the environment in which the train's response to each controller could be tested, so that the values of the fitness function could be calculated over the entire journey.

**Design Choices.** A critical constraint in our design formulation was the cost of running the TSIM simulator. To properly analyze the behavior of the train dynamics we needed to use a very small integration step. As a result, the accurate simulation of a 40-mile track required more than 12-15 seconds on a Sun Sparc10. Since a complete simulation was required during each trial, we wanted to limit the number of trials, which is the product of population size and number of generations, while still achieving a reasonable convergence. To improve the convergence, we decided to limit the chromosome length, which is one of the factors that affect convergence. Therefore, we opted for a sequential tuning approach of scaling factors, membership functions and rule sets. Furthermore, we decided to use a coarse precision for each of the SF by using eight levels (three bits), since that granularity was enough to ensure the FLC's good performance.

There are other promising alternatives that we omitted in this first phase of the project, such as the simultaneous tuning of both scaling factors and membership functions or the use of a finer granularity obtained by increasing the chromosome length. These alternative design choices will be addressed in our future work.

In this experiment, we performed eight tests 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 two fitness functions.

**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 (which captures travel in the plains) and an approximately

40-mile piece of actual Conrail track from Selkirk to Framingham over the Berkshires in Massachusetts (which captures mountainous terrain). The train weighed nearly 9000 tons, and was about one mile long, with four locomotives and 100 fully loaded railcars. An analytically computed velocity profile that 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, the terms were uniformly positioned trapezoids overlapping at a 50% level over the normalized universe of discourse. Since the controller was defined by a nonlinear control surface in  $(e, \Delta e, \Delta u)$  space, we needed three termsets in all, one for each of these three variables. This design leads to a symmetric controller, which is not always a good assumption. In this case, the GA tuning automatically created the required asymmetry.

**GENESIS parameters.** The GA parameters were initialized according to the default values suggested by De Jong [109], i.e.: 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 asymptotic convergence [110], gray codes were used in the encoding, 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).** When tuning membership functions (MFs), a chromosome is formed by concatenating the 21 parameterized MFs. Since each MF is trapezoidal and the overlap degree of 0.5 is maintained between adjacent trapezoids, this partitions the universe of discourse into intervals that alternate between being cores of a MF, and overlap areas. The core of NH and PH extend semi-infinitely to the left and right, respectively (outside the  $[-1,1]$  interval). These intervals, denoted by  $b_i$  are 11 in number for 7 MF labels. Typically,  $\#(b) = 2 \times \#(\text{MF}) - 3$ , where  $\#(b)$  is the number of required intervals and  $\#(\text{MF})$  is the number of membership functions. Each chromosome is thus a vector of 11 floating point values. Since the universe is normalized,  $\sum_{i=1}^{11} b_i \leq 2$ . For the present study, each  $b_i$  is set within the range of  $[0.09; 0.18]$  and five bits are used to represent a chromosome for GA-tuned membership functions. If  $\sum_{i=1}^{11} b_i$  exceeds 2, this implicitly reduces the number of effective MFs, thus providing partial structure optimization as well.

## 5.5 Results

### 5.5.1 Tuning SF vs. MF with Fitness Function $f_1$

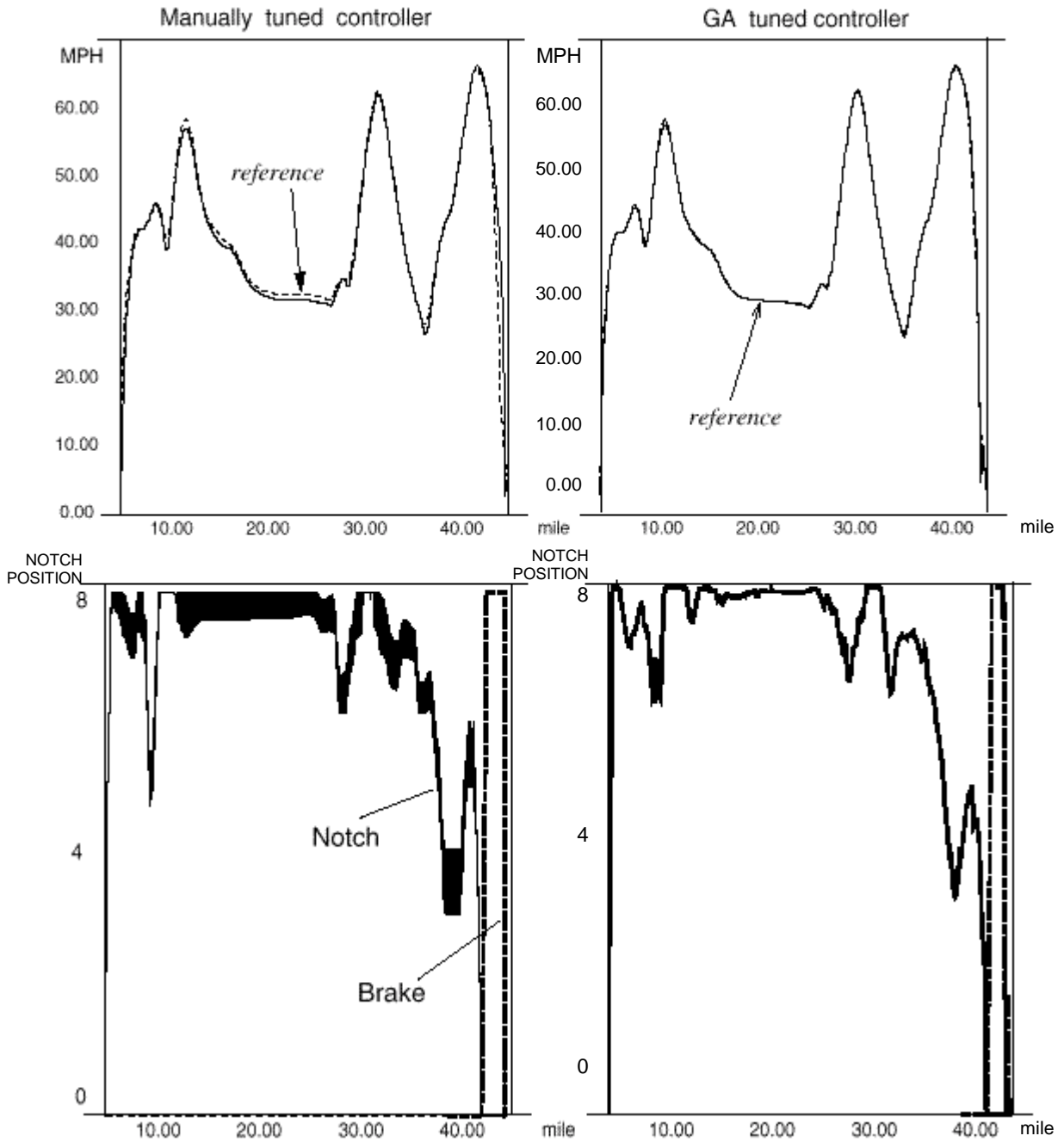
Four sets of tests were conducted for the comparison of significance in tuning scaling factors (SF) versus tuning membership functions (MF) with respect to fitness function  $f_1$ . The results are shown in Table 1. The initial set of scaling factors is  $[S_e S_d S_u] = [5.0; 0.2; 5000]$ . After 4 generations of evolution, the set of GA-tuned SF is  $[9.0; 0.1; 1000]$ . As shown in Table 1, the fitness value was dramatically increased from -73.20 to -15.15 for the GA tuned SF with respect to  $f_1$ . Substantially smoother control is the result.

**Table 1. Summary of simulation results on 14-mile flat track**

Description	f1	f2
Initial SF w/initial MF	- 73.20	- 1.00
GA tuned SF w/initial MF	- 15.15	- 0.82
Initial SF w/GA tuned MF	- 70.93	- 0.94
GA tuned SF w/GA tuned MF	- 14.64	- 0.82

On the other hand, GA-tuned MF only reduced throttle jockeying slightly after 20 generations of evolution. This is reflected in the small increase in fitness value from -73.20 to -70.93. From the above observations, we can conclude that tuning scaling factors is more cost-effective, and has a larger impact than tuning membership functions. The small improvement by tuning MFs leads to an asymmetric shift in the membership functions, such that the ones corresponding to the brake lever are shifted more than the ones corresponding to the throttle setting. 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 learns to respond to negative errors slightly differently than to positive errors.

Next, we demonstrate that tuning membership functions only gives marginal improvements for a fuzzy PI controller with tuned scaling factors. We used GAs to optimize FPI's MFs 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 increased from -15.15 to -14.64. The change in smoothness and the MF values is minimal. After observing the simulation results, we concluded that tuning membership functions alone provided only marginal improvements for a fuzzy PI controller with tuned scaling factors.



**Figure 5. Performance graph with manually tuned and GA tuned controller**

### 5.5.2 Tuning SF vs. MF with Fitness Function $f_2$

We further verified our arguments stated in Section 5.5.1 with the following four sets of tests conducted with respect to fitness function  $f_2$ , which balances both tracking error and control smoothness.

Recall that the initial set of scaling factors is [5.0, 0.2, 5000]. After four generations of evolution, the GA discovered a set of SF = [3.3, 0.9, 5571], with respect to function  $f_2$ . As shown in Table 1, the fitness value was increased from -1.00 to -0.82 while doing this.

On the other hand, GA tuned MF only increased the fitness value from -1.00 to -0.94 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.

Table 1 summarizes all 8 tests run on the 14-mile flat track. In addition, we present the final performance graphs in Figure 5 for the more complex piece of 40-mile real track with the same train. It shows substantial improvement in control accuracy and smoothness. The two figures on the left show reference tracking performance and control outputs before GA tuning. The two figures on the right show vastly improved tracking and smoothness of control.

## 5.6 Summarizing the SC Application to Control

We have presented an approach that uses genetic algorithms to tune fuzzy systems for a complex control problem. We showed that all parameters do not need to be treated as equally important, and that sequential optimization, using genetic algorithms, can greatly reduce computational effort by tuning scaling factors first. Additional improvement was shown by the good performance of fairly coarse encoding. 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 configurations. In this way, we can efficiently produce customized controllers offline. Future work will also focus on automatic generation of velocity profiles for the train simulator by using GAs for trajectory optimization subject to “soft” constraints.

The problem context that we discussed resembles reinforcement learning, since the fitness function is cumulative over many output responses of the controller. The model uses fuzzy rules, but the rule structure is different from the fuzzy clusters used in the previous diagnostics problem. These rules are much more structured, with the standard 3-dimensional FPI control surface, and a large number of degrees of freedom due to the membership functions. The parameter search space is also more complex, but the search method is more sophisticated. A simple filtering technique like the one used before would not work here because of the reinforcement learning issue

and because the filter introduces a lag whereas this problem needs a look-ahead capability. On the other hand, the sequential optimization makes the search practical without sacrificing performance. Finally, the performance requirement is crucial enough that global search methods such as evolutionary algorithms are a better fit than the alternative gradient-descent based techniques. The hybridization is again of a complementary kind, where the EC algorithms can exploit the hierarchy of importance of the parameters in the FPI, unlike a neural network where it is not clear if any weights could be tuned prior to the others.

## 6 Valuing Residential Properties using a Fuzzy-Neural System

### 6.1 Problem Description

Residential property valuation [111] is the process of determining a dollar estimate of the property value for given market conditions. For this paper we restrict ourselves to a single-family residence designed or intended for owner-occupancy. The value of a property changes with market conditions, so any estimate of its value must be periodically updated to reflect those market changes. Any valuation must also be supported by current evidence of market conditions, e.g., recent real estate transactions.

The current manual process for valuing properties usually requires an on-site visit by a human appraiser, takes several days, and costs about \$500 per subject property. This process is too slow and expensive for batch applications such as those used by banks for updating their loan and insurance portfolios, verifying risk profiles of servicing rights, or evaluating default risks for securitized packages of mortgages. The appraisal process for these batch applications is currently estimated, to a lesser degree of accuracy, by sampling techniques. Verification of property value on individual transactions may also be required by secondary buyers and mortgage insurers. Thus, this work is motivated by a broad spectrum of application areas. Some of the applications also require that the output be qualified by a reliability measure and some justification, so that questionable data and unusual circumstances can be flagged for the human who uses the output of the system.

The most common and credible method used by appraisers is the *sales comparison* approach. This method consists of finding comparables, i.e., recent sales that are comparable to the subject property (using sales records); contrasting the subject property with the comparables; adjusting the comparables' sales price to reflect their differences from the subject property (using heuristics and personal experience); and reconciling the comparables adjusted sales prices to derive an estimate for the subject property (using any reasonable averaging method). This process assumes that the item's market value can be derived by the prices demanded by similar items in the same market. To automate the valuation process we have

developed the Automated Residential Property Valuation (ARPV) system, which combines the result of two independent estimators – one using neuro-fuzzy networks, and one using case-based reasoning [112].

First, we will discuss the data used to develop and test our algorithm, followed by a brief overview of the methods employed to build individual estimators. We will then describe in detail a neuro-fuzzy-based estimator. Finally, we will present a method for fusing the output of the estimators into a single estimate and reliability value.

### 6.1.1 Data Description

The database used consists of public sales records of every property that has been sold in certain counties in Northern and Southern California during the five years preceding the development work. Data was purchased from two sources, which we shall name A and B. Data vendors provided a description of each property, which, in some cases, contained up to 166 property attributes. In addition, sales information (such as price and date) was also provided. A subset of the attributes describing a property is given in Table 2.

**Table 2: Subset of Property and Sale Attributes**

Attribute	Value
Sale Price	\$ 175,500
Sale Date	7/23/93
Address1	12 Bronco Lane
City	Contra Costa
State	CA
Zip Code	94063
Living Area	2,200 sq. ft.
Lot Size	10,000 sq. ft.
Bedrooms	3
Bathrooms	2
Total Rooms	6
Age	10 years

## 6.2 AIGEN Estimator

The generative AI model (AIGEN) relies on a fuzzy-neural net that, after a training phase, provides an estimate of the subject’s value.

The specific model that we chose for the AIGEN estimator is a network-based implementation of fuzzy inference. Part of it is based on ANFIS [67], which implements a fuzzy system as a five-layer neural network so that the structure of the net can be interpreted in terms of high-level rules. This net is then trained automatically from data. We developed an extension of this methodology (referred to as E-ANFIS) which we use in AIGEN. ANFIS and E-ANFIS implement the fuzzy reasoning process as a neural net, and rely minimally on expert knowledge and mostly on market data. In essence, the fuzzy logic helps to specify the structural component of the model only.

### 6.2.1 E-ANFIS

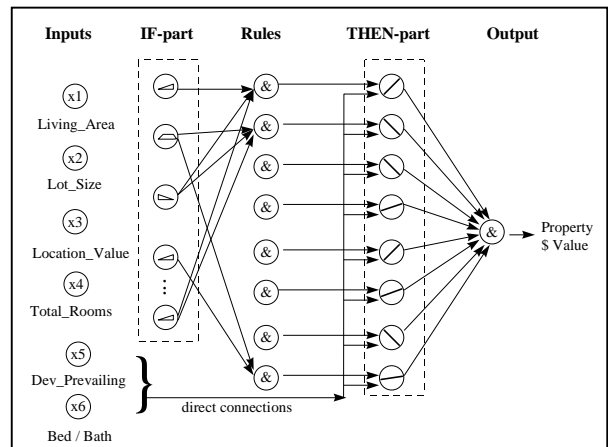
Figure 6 shows the architecture that we used for the automated property valuation problem. The central difference between ANFIS and E-ANFIS is that E-ANFIS allows the output to be linear functions of variables that do not necessarily occur in the input. Another way of stating that is to allow the segmentation of the input space on a proper subset of the total variable set only and then using a cylindrical projection of that segmentation for the whole space.

Figure 7 shows a schematic for the fuzzy inference process where the rules are TSK-type. The semantics are of the form : “if *lot\_size is small* and *living\_area is small* and *locational\_value is high*, then price is *f(...)*”, where *f()* is a linear function of the 6 input variables. A special case of this is when all *c<sub>ij</sub>* except *c<sub>i0</sub>* are 0, in which case each rule recommends a fixed, crisp number. The inference procedure with TSK-type rules yields

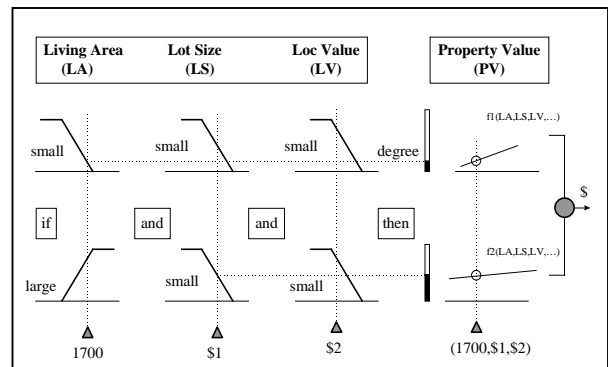
$$z = \sum_i w_i (c_{i0} + c_{i1}x + c_{i2}y) / \sum_i w_i$$

where *w<sub>i</sub>* is the weight of rule *i*, computed as a weighted sum. For the antecedent fuzzy membership functions, we use a generalized bell function (softened trapezoids) given by

$$\mu_A(x) = 1 / (1 + ((x-c) / a)^{2b})$$



**Figure 6: An E-ANFIS network**



**Figure 7: The Fuzzy Inference Process**

The E-ANFIS network architecture is determined by the number of membership functions assigned to each input dimension. For instance, if there are six inputs in all, and two membership functions are assigned to each of four of the inputs, the net has 6 input units, 8 units in the first layer (which come from the two fuzzy functions for each of the four variables), 16 in the next two layers (since  $2 \times 2 \times 2 \times 2 = 16$  rules), and finally one summation unit to produce the output in the output layer. Each of the 16 rules has a TSK-type consequent that depends on all 6 inputs. Since each antecedent membership function has 3 degrees of freedom ( $a$ ,  $b$ , and  $c$ ), and each consequent has 7 coefficients here, there are  $8 \times 3 + 16 \times 7 = 136$  degrees of freedom for this E-ANFIS model.

Once the architecture is constructed in this fashion, the parameters can be initialized with reasonable values, representing current domain knowledge, instead of randomly as in neural networks. For instance, the membership functions can be spaced at uniform distances over the axis so as to cover the range of the data points. The consequent linear functions are initialized to zero.

### 6.2.2 Training E-ANFIS

A variant of the gradient descent technique is used to train the network based on the training data. In brief, this algorithm tries to minimize the mean squared error between the network outputs and the desired answers, when presented with the data points in the training set. It proceeds as follows:

- Present a sample point in the training data set to the network and compute its output.
- Compute the error between the network output and the desired answer.
- Holding the IF-part parameters fixed, solve for the optimal values of the THEN-part parameters using a least-mean-squares optimization method. A recursive Kalman filter method is used here.
- Compute the effect of the IF-part parameters on the error, using derivatives of the functions implemented by intermediate layers.
- Using the above, change the IF-part parameters by small amounts so that the error at the output is reduced.
- Repeat the above steps several times using the entire training set, until the error is sufficiently small.

This procedure is as discussed in [67], with the extension of the Kalman filter to input variables that are not part of the antecedents of the fuzzy rules.

This procedure converges to a locally optimal configuration where the error becomes fixed or decreases very slowly. The resulting network can be interpreted as a fuzzy rulebase, with each parameter in the net having a definite meaning in terms of the fuzzy sets or consequent functions. Learning speed is very fast compared to the conventional neural net paradigm. Additional data, if available, can always be used to further train the model

using the same backpropagation-type algorithm. The resulting surface is well behaved and provably smooth. The rule base is extremely compact, so a large number of models can be stored easily. This is an important consideration, since there are thousands of counties in the US, and each one will need at least one model.

The properties used for training the AIGEN model were restricted to be in a certain price range. This is to eliminate obvious outliers. For this reason, it should not be used to make a prediction on a property outside this range. It will make a prediction if given such a property, but will bound its output to the said range. It will also issue a warning to the user in this case.

### 6.2.3 Constructing AIGEN models

The training algorithm used was the backpropagation-like method described earlier. We reduced the computational cost associated with training AIGEN by using a heuristic to reduce the dimensionality of the parameter space during training.

The 16 rules have 112 degrees of freedom in the consequent. This is a large share of the dimensionality of the parameter space, which uses a variant of the Kalman filtering algorithm to train the parameters in the consequent. It was found easier to train the consequent partially (4 inputs = 80 parameters) in the interleaved back propagation process described earlier, followed by a final batch phase where all 112 consequent parameters were retrained again while holding the antecedent parameters constant. This reduces the computational cost somewhat. This is similar in principle to the sequential optimization idea used for the train control in the previous application.

Recall that the model search process minimizes the mean squared error over a training set at the present time. All the training, models, and results discussed here are with respect to this optimization metric. The choice of optimization criteria is pivotal to the model that is extracted by the process.

**Training set size:** We used about 5% of the total available data for training purposes. For one of our counties, this meant that out of the 37712 potential test records that were available from Data Source A, 1768 records were extracted and used for training the AIGEN model. For another data source, less than half of this (14-15 thousand records) was available, so that 10% of the total data was used for training the models. This corresponds to about 1400-1500 records for training.

These had to be error-free and more or less randomly distributed so as not to bias the estimator. The specific size of the training set is not significant. Since there are 136 degrees of freedom in all, a good rule of thumb is to use at least ten times as many examples for training. Moreover, it is important to use a small part of the total data, so that overfitting to the data can be avoided. The results discussed below apply to this training scheme.

Inputs to the E-ANFIS network are based on seven attributes of the property:

- *total\_rooms*,
- *num\_bedrooms*,
- *num\_baths*,
- *living\_area*,
- *lot\_size*,
- *locational\_value*,
- *deviation\_from\_prevaling\_value*.

The number of bedrooms and bathrooms is combined to produce a *bedrooms/bathrooms ratio* that is fed along with the other five values to the E-ANFIS net. The last two variables are outputs from another estimator, LOCVAl, which is briefly described in Section 7.1.1. These two variables serve as a rough starting point for the AIGEN estimator. Of these six inputs, *total\_rooms*, *living\_area*, *lot\_size*, *locational\_value* are used for partitioning the space into 16 fuzzy regions. The output variable is simply the dollar value of the house.

### 6.3 Results and Validation

Validation was done by testing the model on the entire dataset available from a data source, after it was filtered to remove atypical properties. The filters used for testing were the same as the ones used for screening the training set. Recall that the training metric was mean squared dollar difference between the actual and estimated price. We used the median of absolute relative error as the principal test metric, defined as the median value of  $E = (\text{actual price} - \text{estimate})/\text{actual price}$ . The median was chosen for its robustness to outliers.

The test dataset used for validation consisted of 35370 property records from data source A. The training data size was 1768 points - only 5% of the total data.

The median value of E was in the range of [8.5%, 8.64%]. The average bias of the estimator was about \$557, which implies that the estimator was very slightly biased towards the conservative side, not an undesirable feature in this context. The bias shown by the unsigned relative error was higher, but this was purely because the same dollar value error translates into different relative errors, depending on the value of the property. The average E value was around 12%, since there are some extreme outliers in the test set.

Note that some limited amount of outlier elimination is took place in the model itself, since it bounded its own estimate to lie in the range used for training. Without the truncation, median and mean of E were not significantly affected, and the RMSE increased slightly. This was to be expected, since it attached very high weights to the few far outliers. For test data in the [5%,95%] percentile range of signed relative error, the coefficient of correlation between the actual price and the estimated AIGEN price was 0.97. Additional quality control of the data using some filters (on

both the training and test sets), led to an improved median E of about 7-8%.

When data source B was used (which is of worse quality and completeness), the test set size was 15733 records for the same county and time period. Training was done using 10% of these. The model performance with the same structure and search method was slightly worse, which is not unexpected.

We tried several alternative model structures as well, by varying the input variables fed to the net. The cumulative probability distributions for error E over the test sets (35370 points for A, 15733 for B) were compared. We confirmed that models based on dataset B were worse than the ones based on dataset A, but that the spread between model performances within each of the two sets was not very large (with A again producing more consistent models than B). Changing filters and price ranges improved the performance with respect to data set B as well.

We also did cross-checking experiments where a model trained on data source A was tested on data source B and vice versa. As expected, test performance deteriorated, partly due to data quality and partly due to inconsistent semantics used by the collectors of the two data source companies. This demonstrated the intuitive fact that when the model is used, the inputs to it should not only be accurate and complete, but consistent with the semantics of the training data.

### 6.4 Discussion

We used a complementary hybrid – a neuro-fuzzy system, to solve the property valuation problem in one particular way with AIGEN. Encouraging results were demonstrated in one California county, based on multiple data sources. This was despite the limited number of attributes used in the computation. These models were built for a few other counties as well, with comparable test results.

Considering that AIGEN looked only at five structural variables, and did not consider style, condition, quality of construction, and economic trend metrics, these were encouraging results. The other physical variables were not used since they tended to be missing from the records in a significant portion of the database.

Non-AI approaches such as statistical regression use a similar philosophy of training and testing the models. AIGEN can be thought of as a fuzzy extension of the CART tool [113], which uses an explicit tree structure as the model and the branching thresholds as parameters. The parameters are identified using a greedy variance-minimizing search algorithm. Our approach, in addition to being a smooth combination of piecewise linear regions, provides a transparent explanation of the model in terms of a small number of rules that are comprehensible to a human.

The hybridization in AIGEN is also complementary, where neural techniques help to define a fuzzy structure. As

discussed earlier, small variations in model structure and inputs did not seem to affect the performance much. The number of nominal parameters was in the range of 100-150, but the number of effective parameters is probably less than that. The search method was entirely gradient-based and could conceptually get trapped in a local minimum. However, there is independent evidence (from other estimators) that since this is a real-world illiquid market with noisy prices, there will always be an inherent irreducible error of about 5%, which a global search could not beat. Hence, using a local search technique here was much more practical than using genetic or evolutionary algorithms for search.

## 7 Residential Property Valuation Using a Fusion of Hybrid Soft Computing Models

In Section 6 we focused on using a complementary hybrid of two SC technologies to solve a particular problem. If two or more such models could be built independently for the same problem, which one should be chosen? We propose that a loose hybrid, which could be called information fusion or model fusion, is a much more robust alternative to picking one of the models. We demonstrate this with the same application of residential property valuation where multiple estimators, based on location (LOCVAL), comparable properties (AICOMP), and a generative model (AIGEN), were built using different technologies on the same database. This combines the results of the models rather than the techniques themselves, and the focus is on using the redundancy to increase reliability rather than accuracy alone.

### 7.1 Estimators Used in Fusion

#### 7.1.1 Locational Value Model (LOCVAL)

The first model was based solely on the location and living area of the properties, as shown in Figure 8.

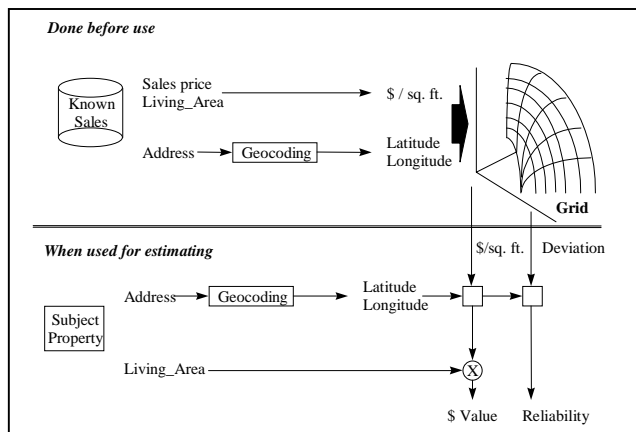


Figure 8: Locational Value method (LOCVAL)

A dollar per square foot measure was constructed for each point in the county, by suitably averaging the

observed, filtered historical market values in the vicinity of that point. This locational value estimator (LOCVAL) produces two output values: *Locational\_Value* (a \$/sq.ft. estimate) and *Deviation\_from\_prevaling\_value*. The local averaging is done by an exponentially decreasing radial basis function with a “space constant” of 0.15-0.2 miles. It can be described as the weighted sum of radial basis functions (all of the same width), each situated at the site of a sale within the past 1 year and having an amplitude equal to the sales price. Deviation from prevailing value is the standard deviation for houses within the area covered and is derived using a similar approach.

In order to use the LOCVAL estimator correctly, the input values (a valid, geocoded address and a living area in squared feet) must be present and accurate for the locational estimator to work correctly. If either is missing, or clearly out-of-range, the estimator will not make any prediction.

#### 7.1.2 Comparable Value Model (AICOMP)

The second model, AICOMP, relied on a case based reasoning (CBR) process similar to the sales comparison approach [111] used by certified appraisers to estimate a residential property's value. The CBR process consists of selecting relevant cases (which would be nearby house sales), adapting them, and aggregating those adapted cases into a single estimate of the property value.

Gonzalez first showed the possibility of using CBR to automating the valuation process [114]. However, his CBR approach never captured the intrinsic imprecision of the basic steps in the sale comparison approach: finding the *most similar* houses, located *close* to the subject property, sold *not too long* ago; and selecting a *balanced* subset of the *most promising* comparables to derive the final estimate. Therefore we developed AICOMP, a fuzzy CBR system that uses fuzzy predicates and fuzzy-logic based similarity measures [115] to estimate the value of residential property. Our approach, shown in Figure 9 and further described in [3], consists of:

- 1) *Retrieving recent sales from a case-base.* Upon entering the subject property attributes, AICOMP retrieves potentially similar comparables from the case-base. This initial selection uses six attributes: address, date of sale, living area, lot area, number of bathrooms, and bedrooms.
- 2) *Comparing the subject property with the retrieved cases.* The comparables are rated and ranked on a similarity scale to identify the most similar ones to the subject property. This rating is obtained from a weighted aggregation of the decision maker preferences, expressed as fuzzy membership distributions and relations.
- 3) *Adjusting the sales price of the retrieved cases.* Each property's sales price is adjusted to reflect their differences from the subject property. These adjustments are performed by a rule set that uses

additional property attributes, such as construction quality, conditions, pools, fireplaces, etc.

- 4) *Aggregating the adjusted sales prices of the retrieved cases.* The best four to eight comparables are selected. The adjusted sales price and similarity of the selected properties are combined to produce an estimate of the subject value with an associated reliability value.

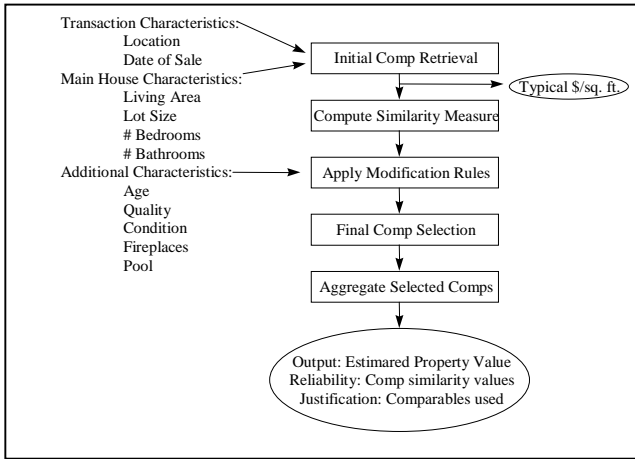


Figure 9: CBR Process

### 7.1.3 Generative Method (AIGEN)

The third method, called AIGEN, is a generative AI method in which a fuzzy-neural net is trained by using a subset of cases from the case-base. The resulting run-time system provides an estimate of the subject's value. This method was described in Section 6.

### 7.1.4 Model Outputs: Property and Reliability Values

Each model produced a property value and an associated reliability value. The latter was a function of the "averageness" or "typicality" of the subject property based on its physical characteristics (such as lot size, living area, total room). These typical values were represented by possibilistic distributions (fuzzy sets). We computed the degree to which each property satisfied each criterion. The overall property value reliability was obtained by considering the conjunction of these constraint satisfactions (i.e., the minimum of the individual reliability values). A more detailed description of this process can be found in [116].

The computation times, required inputs, errors and reliability values for these three methods are shown in Figure 10. The locational value method takes the least time and information, but produces the largest error. The CBR approach takes the largest time and number of inputs, but produces the lowest error.

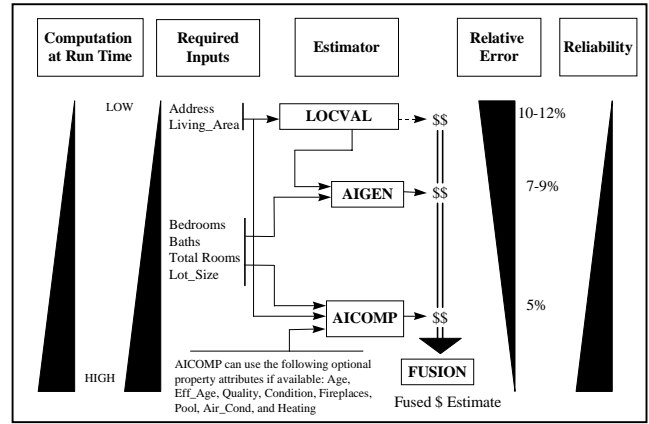


Figure 10: Data comparison of multiple approaches

### 7.1.5 Advantages of Fusion

The fusion of the three estimators has several advantages:

- the fusion process provides an indication of the reliability or trust in the final estimate,
- if reliability is high, the fused estimate is more accurate than any of the individual ones
- if reliability is limited, the system generates an explanation in human terms, and
- the fused estimate is more robust.

These characteristics allow the user to determine the suitability of the estimate within the given business application context. Knowledge-based rules are used for constructing this fusion of technologies at a supervisory level, and the parameters are few enough to be determined by some inspection and experimentation.

### 7.2 Related Work

Combining forecasts and estimates has been extensively studied in disciplines such as econometrics and market research [117-121]. A common fusion technique is to use a weighted average where the weights are related to some measure of reliability in the individual forecasts. This is essentially the approach we have taken in this work. On the other hand, we believe the method we have used to obtain a combined reliability is unique.

### 7.3 Approach to Fusion

Our approach is broken down into four steps.

- 1) Measure the reliability or "goodness" of each individual estimate. This is done individually by each estimator, using a fuzzy definition of "typicality".
- 2) Combine the three \$ values produced by the individual estimators into a fused \$ value.
- 3) Use a rule-base that determines the overall quality of the fused estimate -- this is referred to as the fusion of the reliability values.
- 4) Generate explanatory messages to the user when the overall quality is limited. These messages enable the user to understand why the best possible estimate

cannot be produced. Further information by the user may then enable the system to improve its results.

We will refer to the overall system as ARPV (Automated Residential Property Valuation). The final output of the system will be referred to as the *ARPV-Estimate*. The accompanying measure of reliability is called the *Quality* of this estimate.

## 7.4 Fusion Rules

As seen in Figure 10, LOCVAL is used as an input to AIGEN, and AICOMP and AIGEN both use property descriptions to produce the answers. These two are then combined to produce the final *ARPV-Estimate* of property value and an overall measure of reliability or *Quality*. Finally, ARPV provides a description of circumstances for the data, property, and environment that may have led to a limited value of *Quality*.

The precise manner of their combination depends on the relative strengths of the two individual reliability values, and the amount of disagreement between the two estimator values. Each reliability value is a number in  $[0,1]$ . The amount of disagreement is measured by *contention* between the two values  $v_{AIG}$  (produced by AIGEN), and  $v_{AIC}$  (produced by AICOMP). Contention  $c$  is computed as

$$c = (v_{AIG} - v_{AIC}) / ((v_{AIG} + v_{AIC}) / 2)$$

and is defined only when both estimators have produced a valid answer.

*ARPV-Estimate*, the fused property value, takes one of the following values:

- The mean of the two estimates, weighted by their respective normalized reliability values – if both AIGEN and AICOMP estimates are known
- The value of the available estimator – if one of AIGEN and AICOMP estimates is known but the other is not, making their combination impossible.
- The value of LOCVAL – if both AIGEN and AICOMP are unknown, making their combination impossible

*Quality*, the measure of the reliability of the fused ARPV estimate, takes one of the following three values:

- EXCELLENT -- All essential data are available, reliability for the independent estimators is high, and the independent estimators agree.
- INDICATIVE -- All essential data are available, reliability of one or more estimators is not high due to somewhat unusual property and/or local market characteristics, and/or the estimators disagree.
- UNRELIABLE -- Some essential data may be missing, and/or reliability in one or more estimators is low due to very unusual property and/or local market characteristics, and/or the estimators disagree markedly.

A more detailed description of the fusion rules can be found in Bonissone et al. 1998.

## 7.5 Fusion Results

The reliability value generated by the fusion rules was subdivided into three groupings (*good*, *fair*, and *poor*). The reliability measure should then produce the largest good set with the lowest error.

From a test sample of 7,293 subjects, we could classify our reliability in 63% as *good*. The *good* set had a medium absolute error of 5.4%, an error that was satisfactory for the intended application. Of the remaining subjects, 24% were classified as *fair*, and 13% as *poor*. The fair set had a medium error of 7.7%, and the poor set had a median error of 11.8%.

## 7.6 Summarizing the SC Applications to Property Valuation

We have considered the important and difficult problem of residential property valuation and shown that techniques based on soft computing can successfully solve it. A multitude of approaches using fuzzy logic, case-based reasoning, and neural networks (in hybrid combination) was shown to be useful in this regard. Moreover, the reliability computation and the fusion process increase the robustness and human usefulness of the system. It has been shown to achieve good accuracy and be scaleable for thousands of automated transactions. This makes it a transparent, interpretable, fast, and inexpensive choice for bulk estimates of residential property value for a variety of financial applications.

## 8. Final Remarks

Soft computing (SC) is having an impact on many industrial and commercial operations, from predictive modeling to diagnostics and control. It provides us with alternative approaches to traditional knowledge-driven reasoning systems or pure data-driven systems (the fundamental problems of these classical approaches are discussed in Section 1.4) and it overcomes their shortcomings by synthesizing a number of complementary reasoning and searching methods over a large spectrum of problem domains. We have reviewed soft computing's main components (fuzzy logic, probabilistic reasoning, neural networks, and evolutionary computing) and surveyed some of their successful combinations that have led to the development of hybrid SC systems.

These systems leverage the tolerance for imprecision, uncertainty, and incompleteness, which is intrinsic to the problems to be solved, and generate tractable, low-cost, robust solutions to such problems. The synergy derived from these hybrid systems stems from the relative ease with which we can translate problem domain knowledge into initial model structures whose parameters are further tuned by local or global search methods. This is a form of

complementary or “tight” hybridization. Apart from this type of hybridization, we also discussed the fusion of estimators – this type of model fusion or “loose” hybridization does not combine features of the methodologies themselves, but only their results. The primary motivation here is to increase reliability rather than to make model construction easier.

We have illustrated this synergy by describing three applications in diagnostics, control, and prediction, as summarized in Table 3. These studies stem from real-world, high-impact business problems, such as gas turbine service and diagnosis, freight train control, and residential property valuation, respectively, and use a number of different means from the toolbox of soft computing. In particular, all approaches used a hybridization of SC techniques to overcome hurdles posed by the problem, such as high noise and low information content of sensor data, high system complexity, and high-dimensional feature space.

The payoff of this conjunctive use of techniques is a more accurate and robust solution than a solution derived from the use of any single technique alone. This synergy comes at comparatively little expense because typically the

problem, as was evidenced by the property estimation application. If there are several possibilities for the structure and the search methods, many more pairings of technologies are possible, and problem solving becomes easier. For instance, AIGEN used a fuzzy model with local gradient search, but one could also use a pure neural network trained by an evolutionary algorithm. Of course, computation time, cost, business needs, and data requirements will then influence the choice of the combination of technologies. A step in further improving system performance is the exploitation of parallel systems. These systems may be designed to rely to the maximum amount on non-overlapping data and use different techniques to arrive at their conclusions. In *information fusion*, the outputs of these heterogeneous models will be compared, contrasted, and aggregated, as seen in our last application.

The future appears to hold a lot of promise for the novel use and combinations of SC applications. The circle of SC's related technologies will probably widen beyond its current constituents. The push for low-cost solutions combined with the need for intelligent tools will result in the deployment of hybrid systems that efficiently integrate

**Table 3. Soft computing applications in industry and business discussed in this paper.**

Application Area	Problem Type	Type of Hybrid	Model Structure Type	Parameter Search Method
Industrial equipment anomaly detection	Diagnostics	Complementary	Fuzzy clusters in high-dimensional space	Exponential Weighted Moving Average Filter
Automatic train handling	Nonlinear Control	Complementary	Fuzzy PI Controller rule-base	Genetic algorithms
Residential Property Valuation	Predictive Modeling	Complementary	Fuzzy if-then rules as network	Variation on back propagation / gradient descent
Residential Property valuation	Predictive Modeling	Fusion	Rule-base for fusing result & reliability	Manually determined

methods do not try to solve the same problem in parallel but they do it in a mutually complementary fashion. Another way to say this is that the model needs a structure and parameters, and a search method to discover them, and no single technique should be expected to be the best for all problems. For example, in our control application, a hierarchical fuzzy controller was used to embody the qualitative knowledge used by locomotive engineers to manually perform the task, but manual tuning of this controller would have been an inferior solution. Fast tuning of this controller was obtained by a global search approach, a genetic algorithm that yielded a robust controller whose parameters did not have to be specialized for each particular initial condition.

Another advantage to the hybridization of techniques is that it is easier to think of alternative solutions to the same

reasoning and search techniques.

On the application front we will likely see a drive towards prognostic and autonomous capabilities. With an increase of service-related operations, it will be increasingly attractive to be able to forecast anomalous trends and conditions, and correct them before their effects are fully developed. In addition, remotely monitored systems will bear the need to operate autonomously, thus requiring intelligent agents to regulate their operations.

In the future, we expect that the combination of soft computing with advances in the areas of computer vision, voice recognition, natural language processing, etc., will further improve and expand our problem-solving capability to a large spectrum of industrial and commercial problems.

## 9. References

- [1] P.P. Bonissone, V. Badami, K.H. Chiang,, P.S. Khedkar, K. Marcelle, M.J. Schutten, "Industrial Applications of Fuzzy Logic at General Electric", in *Proc. of the IEEE*, vol. 83, no. 3, pp. 450-465, IEEE, 1995.
- [2] Y-T Chen and P.P. Bonissone, "Industrial Applications of Neural Networks at General Electric," Technical Information Series, 98CRD79, General Electric CRD, Schenectady, NY, October 1998.
- [3] P.P. Bonissone and W. Cheetham. "Financial Applications of Fuzzy Case-Based Reasoning to Residential Property Valuation," in *Proc. Sixth Int. Conf. On Fuzzy Systems (FUZZ-IEEE'97)*, pp. 37-44, Barcelona, Spain, 1997.
- [4] L.A. Zadeh, "Fuzzy Logic and Soft Computing: Issues, Contentions and Perspectives," in *Proc. of IIZUKA'94: Third Int. Conf. on Fuzzy Logic, Neural Nets and Soft Computing*, pp. 1-2, Iizuka, Japan, 1994.
- [5] L.A. Zadeh, "Some reflection on soft computing, granular computing and their roles in the conception, design and utilization of information/intelligent systems," *Soft Computing A Fusion of Foundations, Methodologies and Applications*, vol. 2, no. 1, pp. 23-25, 1998.
- [6] D. Dubois and H. Prade, "Soft computing, fuzzy logic, and Artificial Intelligence," *Soft Computing: A Fusion of Foundations, Methodologies and Applications*, vol. 2, no. 1, pp. 7-11, 1998.
- [7] B. Bouchon-Meunier, R. Yager, and L.A. Zadeh, *Fuzzy Logic and Soft Computing*. World Scientific, Singapore, 1995.
- [8] P.P. Bonissone, "Soft Computing: the Convergence of Emerging Reasoning Technologies," *Soft Computing A Fusion of Foundations, Methodologies and Applications*, vol. 1, no. 1, pp. 6-18, 1997.
- [9] N. Rescher, *Many-valued Logic*, McGraw-Hill, New York, NY, 1969.
- [10] M. Black, "Vagueness: an Exercise in Logical Analysis," *Phil.Sci.* vol. 4., pp-427-455, 1937.
- [11] L.A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp.338-353, 1965.
- [12] J. Lukasiewicz, *Elementy Logiki Matematycznej [Elements of Mathematical Logic]*, Warsaw, Poland: Panstwowe Wydawnictwo Naukowe, 1929.
- [13] L.A. Zadeh, "Foreword," in *Handbook of Fuzzy Computation*, E.H. Ruspini, P.P. Bonissone, and W. Pedycz, Eds., Bristol, UK: Institute of Physics, 1998.
- [14] E.H. Ruspini, P.P. Bonissone, and W. Pedycz, *Handbook of Fuzzy Computation*, Bristol, UK: Institute of Physics, 1998.
- [15] Y-M. Pok and J-X. Xu, "Why is Fuzzy Control Robust," in *Proc. Third IEEE Intl. Conf. on Fuzzy Systems (FUZZ-IEEE'94)*, pp. 1018-1022, Orlando, FL, 1994.
- [16] T. Bayes, "An essay towards solving a problem in the doctrine of chances," *Philosophical Trans. of the Royal Society of London*, vol. 53, pp. 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*, vol. 25, pp. 293--215, 1970.
- [17] E. Shortliffe and B. Buchanan, "A Model of Inexact Reasoning in Medicine," *Mathematical Biosciences*, vol. 23, no. 3-4, pp. 351-379, 1975.
- [18] R. Duda, P. Hart, and N. Nilsson, "Subjective Bayesian Methods for Rule-Based Inference Systems," in *Proc. AFIPS* vol.45, pp. 1075-1082, New York, NY: AFIPS Press, 1976.
- [19] J. Pearl, "Reverend Bayes on Inference Engines: a Distributed Hierarchical Approach," in *Proc. 2nd Natl. Conf. on Artificial Intelligence*, pp. 133-136, Menlo Park, CA: AAAI, 1982.
- [20] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Mateo, CA: Morgan-Kaufmann, 1988.
- [21] J. Kim and J. Pearl, "A Computational Model for Causal and Diagnostic Reasoning in Inference Engines", in *Proc. Eighth Int. Joint Conf. on Artificial Intelligence*, pp. 190-193, Karlsruhe, Germany, 1983.
- [22] A. P. Dempster, "Upper and lower probabilities induced by a multivalued mapping," *Annals of Mathematical Statistics*, vol. 38, pp. 325-339, 1967.
- [23] G. Shafer, *A Mathematical Theory of Evidence*. Princeton, NJ: Princeton University Press, 1976.
- [24] L.A. Zadeh, "Probability Measures of Fuzzy Events," *J. Math. Analysis and Appl.*, vol. 10, pp. 421-427, 1968.
- [25] Ph. Smets, "The Degree of Belief in a Fuzzy Set," *Information Science*, vol. 25, pp. 1-19, 1981.

- [26] W.S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bull Math Biophysics*, vol. 5, pp. 115-133, 1943.
- [27] F. Rosenblatt, "The perceptron, a Perceiving and Recognizing Automaton," Project PARA, Cornell Aeronautical Lab. Rep., no. 85-640-1, Buffalo, NY, 1957.
- [28] F. Rosenblatt, "Two theorems of statistical separability in the perceptron," in *Proc. Mechanization of Thought Processes*, pp. 421-456, Symposium held at the National Physical Laboratory, HM Stationary Office, London, 1959.
- [29] F. Rosenblatt, *Principle of Neurodynamics: Perceptron and the theory of Brain Mechanisms*, Washington, DC: Spartan Books, 1962.
- [30] M. Minsky and S. Papert, *Perceptrons*, Boston, MA: MIT Press, 1969.
- [31] P. Werbos, *Beyond Regression: New Tools for Predictions and Analysis in the Behavioral Science*. Ph.D. thesis, Harvard University, Cambridge, MA, 1974.
- [32] D. Parker, "Learning Logic," Tech. Report TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA, 1985.
- [33] Y. LeCun, "Une procedure d'apprentissage pour reseau a seuil symetrique," *Cognitiva*, 85, pp. 599-604, CESTA, Paris, France, 1985.
- [34] K. Hornick, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359-366, 1989.
- [35] J. Moody and C. Darken, "Fast learning in networks of locally tuned processing units," *Neural Computation*, vol. 1, no. 2, pp. 281-294, 1989.
- [36] T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps," *Biological Cybernetics*, vol. 43, pp. 59-69, 1982.
- [37] J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," in *Proc. Acad. Sci.*, vol. 79, pp. 2554-2558, 1982.
- [38] A. Carpenter and S. Grossberg, "A Massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer, Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115, 1983.
- [39] R. Jang, C-T. J. Sun and C. Darken, "Functional Equivalence Between Radial Basis Function Networks and Fuzzy Inference Systems," *IEEE Trans. on Neural Networks*, vol. 4(1), pp. 156-159, 1993
- [40] E. Fiesler, and R. Beale, *Handbook of Neural Computation*, Bristol, UK: Institute of Physics, and New York, NY: Oxford University Press, 1997.
- [41] I. Rechenberg, "Cybernetic Solution Path of an Experimental Problem," *Royal Aircraft Establishment*, Library Translation no. 1122, 1965.
- [42] H-P. Schwefel, *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Stromungstechnik*, Diploma Thesis Technical University of Berlin, Germany, 1965.
- [43] L.J. Fogel, "Autonomous Automata," *Industrial Research*, vol. 4, pp. 14-19, 1962.
- [44] L.J. Fogel, A.J. Owens, and M.J. Walsh, *Artificial Intelligence through Simulated Evolution*, New York, NY: John Wiley, 1966.
- [45] A.S. Fraser, "Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction," *Australian J. of Biological Sci.*, Vol 10, pp. 484-491, 1957.
- [46] H. Bremermann, "The Evolution of Intelligence. The Nervous System as a Model of its Environment," Technical Report no. 1 Contract no. 477(17), Dept. of Mathematics, University of Washington, Seattle, 1958.
- [47] J. Reed, R. Tooms, and N. Baricelli, "Simulation of Biological Evolution and Machine Learning," *J. Theo. Biol.*, vol. 17, pp. 319-342, 1967.
- [48] J.H. Holland, "Outline of a Logical Theory of Adaptive Systems," *J. ACM*, vol. 9, pp. 297-314, 1962.
- [49] J.H. Holland, "Nonlinear Environments Permitting Efficient Adaptation," *Computer and Information Science II*, New York, NY: Academic Press, 1967.
- [50] J.H. Holland, *Adaptation in Natural and Artificial Systems*, Cambridge, MA: MIT Press, 1975.
- [51] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press, 1992.
- [52] R.M. Friedberg, "A Learning Machine: Part I.," *IBM Journal of Research and Development*, vol. 3, pp. 282-287, 1958.
- [53] N.A. Barricelli, "Esempi Numerici di Processi di Evoluzione," *Methodos*, pp.45-68, 1954.

- [54] D.B. Fogel, *Evolutionary Computation*. New York, NY: IEEE Press, 1995.
- [55] T. Back, D.B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*, Bristol, UK: Institute of Physics, and New York, NY: Oxford University Press, 1997.
- [56] D.B. Fogel, *The Fossil Record*, New York, NY: IEEE Press, 1998.
- [57] I. Rechenberg, *Evolutionsstrategien: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Fromman-Holzboog Verlag, Stuttgart, Germany, 1973.
- [58] H.-P. Schwefel, *Numerical Optimization of Computer Models*, Chichester: John Wiley, 1981.
- [59] P. J. Angeline, G.M. Saunders, and J.B. Pollack, "An Evolutionary Algorithm that Constructs Recurrent Neural Networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 54-65, 1994.
- [60] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, New York, NY: Springer-Verlag, 1994.
- [61] K. Forbus, "Qualitative Reasoning about Physical Processes," in *Proc. 7th Int. Joint Conf. on Artificial Intelligence*, Karlsruhe, Germany, 1981.
- [62] B. Kuipers, "Commonsense Reasoning about Causality: Deriving Behavior from Structure," in *Qualitative Reasoning about Physical Systems*, D. Bobrow, Ed., pp. 169-203, Cambridge, MA: MIT Press, 1985.
- [63] E.H. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int. J. Man Machine Studies*, vol. 7, no. 1, pp. 1-13, 1975.
- [64] T. Takagi and M. Sugeno, "Fuzzy Identification of Systems and Its Applications to Modeling and Control," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 15, pp. 116-132, Jan-Feb, 1985.
- [65] R. Babuska, R. Jager, and H.B. Verbruggen, "Interpolation Issues in Sugeno-Takagi Reasoning," in *Proc. Third IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'94)*, pp. 859-863, Orlando, FL, 1994.
- [66] H. Bersini, G. Bontempi, and C. Decaestecker, "Comparing RBF and fuzzy inference systems on theoretical and practical basis," in *Proc. of Int. Conf. on Artificial Neural Networks. ICANN '95, Paris, France*, vol.1, pp. 169-74, 1995.
- [67] J.S.R. Jang, "ANFIS: Adaptive-network-based-fuzzy-inference-system," *IEEE Trans. on Systems, Man, and Cybernetics*, 233, pp. 665-685, May-June, 1993.
- [68] O. Cordon, H. Herrera, and M. Lozano, "A classified review on the combination fuzzy logic-genetic algorithms bibliography", Tech. Report 95129, URL:<http://decsai.ugr.s/~herrera/flga.html>, Department of Computer Science and AI, Universidad de Granada, Granada, Spain, 1995.
- [69] C.L. Karr, "Design of an adaptive fuzzy logic controller using genetic algorithms," in *Proc. Int. Conf. on Genetic Algorithms (ICGA'91)*, pp. 450-456, San Diego, CA., 1991.
- [70] M.A. Lee, and H. Tagaki, "Dynamic control of genetic algorithm using fuzzy logic techniques," in *Proc. Fifth Int. Conf. on Genetic Algorithms*, pp. 76-83. Morgan Kaufmann, CA. 1993.
- [71] 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*, pp. 1097-1104, Aachen, Germany, 1993.
- [72] 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)*, pp. 28-33, Orlando, FL., 1994.
- [73] D. Burkhardt and P.P. Bonissone "Automated Fuzzy Knowledge Base Generation and Tuning," in *Proc First IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'92)*, pp. 179-188, San Diego, CA., 1992.
- [74] P.P. Bonissone, P.S. Khedkar, and Y-T Chen, "Genetic Algorithms for automated tuning of fuzzy controllers, A transportation Application," in *Proc. Fifth IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'96)*, pp. 674-680, New Orleans, LA., 1996.
- [75] L. Zheng, "A Practical Guide to Tune Proportional and Integral (PI) Like Fuzzy Controllers," in *Proc First IEEE Int. Conf. on Fuzzy Systems, (FUZZ-IEEE'92)*, pp. 633-640, S. Diego, CA, 1992.
- [76] V.W. Port, "Overview of Evolutionary Computation as a Mechanism for Solving Neural System Design Problems," D2.1 in *Handbook of Neural Computation*, E. Fiesler and R. Beale, Eds., Bristol, UK: Institute of Physics, and New York, NY: Oxford University Press, 1997.
- [77] E. Vonk. L.C. Jain, and R.P. Johnson, *Automatic Generation of Neural Network Architecture Using Evolutionary Computation*. Singapore: World Scientific Publ. Co., 1997.

- [78] X. Yao "Evolving Artificial Neural Networks, " - In this issue, 1999.
- [79] M. Jurick, "Back Error Propagation: A Critique," *IEEE COMPCON 88*, pp. 387-392, San Francisco, CA, 1988.
- [80] D.J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. of the Eleventh Int. Joint Conf. on Artificial Intelligence (IJCAI-89)*, vol.1, pp. 762-767, San Francisco, CA: Morgan Kaufmann, 1989.
- [81] H. Kitano, "Empirical studies on the speed of convergence of neural network training using genetic algorithms," in *Proc. Eighth National Conf. on Artificial Intelligence (AAAI-90)*, vol.2, pp. 789-95, 1990.
- [82] M. McInerney, A.P. Dhawan, "Use of genetic algorithms with backpropagation in training of feedforward neural networks," in *Proc. of 1993 IEEE Int. Conf. on Neural Networks (ICNN '93)*, San Francisco, CA, vol.1, pp. 203-8, 1993.
- [83] V. Maniezzo, "Genetic Evolution of the Topology and Weight Distribution of Neural Networks, " *IEEE Transaction of Neural Networks*, vol. 5, pp. 39-53, 1994.
- [84] S. Harp, T. Samad, and A. Guha, "Toward the Genetic Synthesis of Neural Networks, " in *Proc. Third Int. Conf. on Genetic Algorithms*, pp. 360-369, 1989.
- [85] E. Alba, J.F. Aldana, and J.M. Troya, "Genetic Algorithms as Heuristics for Optimizing ANN Design," in *Proc. Int. Conf. on Artificial Neural Nets and Genetic Algorithms (ANNGA'93)*, pp. 683-689, Innsbruck, Austria, 1993.
- [86] H. Kitano, "Designing Neural Networks Using Genetic Algorithms with Graph generation System," *Complex Systems*, vol. 4, no. 4, pp. 461-476, 1990.
- [87] A.A. Siddiqi, and S.M. Lucas, "A Comparison of Matrix Rewriting versus Direct Encoding for Evolving Neural Networks," in *Proc. IEEE World Congress on Computational Intelligence (WCCI'98)*, pp. 392-397, Anchorage, Alaska, 1998.
- [88] F. Gruau, "Genetic Synthesis of Boolean Neural Networks wit a Cell Rewriting Developmental Process," in *Proc. Second Int. Conf. on Genetic Algorithms (COGAN'92)*, pp. 55-74. Baltimore, MD, 1992
- [89] J. Koza, and J.P. Rice, "Genetic Generation of both the Weights and Architecture for a Neural Network," in *Proc. IEEE Int. Joint Conf. on Neural Networks*, pp. 397-404, 1991.
- [90] D.B. Fogel, L.J. Fogel, and V.W. Porto, "Evolutionary Methods for Training Neural Networks," in *Proc. IEEE Conf. on Neural Networks for Oceanography Engineering*, Washington, DC, pp. 317-327, 1991
- [91] J.R. McDonnell and D. Waagen, "Evolving Neural Network Connectivity," in *Proc. IEEE ICNN'93*, pp. 863-868, San Francisco, CA, 1993.
- [92] D.E. Goldberg, "Genetic Algorithms and Walsh Functions: Part 2, Deception and Analysis," *Complex Systems*, vol. 3, pp. 153-171, 1989.
- [93] D.B. Fogel, *Evolving Artificial Intelligence*, Ph.D. Dissertation, University of California San Diego, CA, 1992.
- [94] C-H. Lee and J-H. Kim, "Evolutionary Ordered Neural Network with a Linked-list Encoding Scheme," in *Proc. IEEE ICEC'96*, pp. 665-669, 1996.
- [95] R.A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, 1, pp. 295-307, 1988.
- [96] P. Arabshahi, J.J. Choi, R.J. Marks, and T.P. Caudell, "Fuzzy Control of Backpropagation," in *Proc. First IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'92)*, pp. 967-972, San Diego, CA., 1992.
- [97] P.D. Wasserman, *Neural Computing: Theory and Practice*. Van Nostrand Reinhold, 1989.
- [98] F. Herrera and M. Lozano, "Adaptive Genetic Algorithms Based on Fuzzy Techniques," in *Proc. of IPMU'96*, pp. 775-780, Granada, Spain, 1996
- [99] R. Subbu, A. Anderson, and P.P. Bonissone, "Fuzzy Logic Controlled Genetic Algorithms versus Tuned Genetic Algorithms: An Agile Manufacturing Application," in *Proc. IEEE Int. Symposium on Intelligent Control*, NIST, Gaithersburg, Maryland, 1998.
- [100] D. Doel, "The Role for Expert Systems in Commercial Gas Turbine Engine Monitoring," in *Proc. of the Gas Turbine and Aeroengine Congress and Exposition*, Brussels, Belgium, 1990.
- [101] S. Karamouzis and S. Feyock, "A Performance Assessment of a Case-Based Diagnostic System for Aircraft Malfunctions," in *Proc. of the Sixth Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Edinburgh, Scotland, pp. 71-78, 1993.
- [102] R. Reibling and S. Bublin, "Diagnostic Reasoning Technology for the On-Board Maintenance System," in *Proc. IEEE 1993 National Aerospace and Electronics Conf. NAECON 1993*. vol. 2, pp. 930-936, 1993.

- [103] M. Fernandez-Montesinos, P. Janssens, and R. Vingerhoeds, "Enhancing Aircraft Engine Condition Monitoring. Safety, Reliability and Applications of Emerging Intelligent Control Technologies," A Postprint Volume from the *IFAC Workshop on Emerging Intelligent Control Technologies*, Hong Kong, pp. 161-166, 1994.
- [104] R. Records and J. Choi, "Spurious Symptom Reduction in Fault Monitoring Using a Neural Network and Knowledge as Hybrid System," in *Proc. of the Twelfth National Conf. on Artificial Intelligence*, vol. 2, Seattle, WA, 1994.
- [105] J.B. Gomm, "Process-Fault Diagnosis Using a Self-Adaptive Neural Network with On-Line Learning Capabilities," in *Proc. IFAC On-Line Fault Detection and Supervision in the Chemical Process Industry*, pp. 69-74, 1995.
- [106] V.C. Patel, V. Kadirkamanathan, and H.A. Thompson, "A Novel Self-Learning Fault Detection System for Gas Turbine Engines," in *Proc. UKACC Int. Conf. on Control '96*, pp. 867-872, 1996.
- [107] E.H. Ruspini, "A New Approach to Clustering. *Information Control*," vol. 15, no. 1, pp. 22-32, 1969.
- [108] J.J. Grefenstette, "GENESIS: A system for Using Search Procedures," in *Proc. of the 1984 Conf. on Intelligent Systems and Machines*, pp. 161-165, 1984.
- [109] K.A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. Thesis, University of Michigan, Ann Arbor, MI. Also available as Dissertation Abstract International, 36(10), 5140B, University Microfilms no. 76-9381, 1975.
- [110] G. Rudolph, "Convergence of Evolutionary Algorithms in General Search Spaces," in *Proc. of the IEEE Int. Conf. on Evolutionary Computation*, Nagoya, Japan, pp. 50-54, 1996.
- [111] Appraisal Institute, *Appraising Residential Properties, Part VI*, Chicago IL, 1994.
- [112] J. Kolodner, *Case-based Reasoning*. San Francisco, CA: Morgan Kaufmann, 1993.
- [113] Breiman, Friedman, Olshen, and Stone, *Classification and Regression Trees*, Monterey, CA: Wadsworth and Brooks, 1985.
- [114] A.J. Gonzalez, "A Case-Based Reasoning Approach to Real Estate Property Appraisal," *Expert Systems with Applications*, vol. 4, no. 2, pp. 229-246, 1992.
- [115] P.P. Bonissone and S. Ayub, "Similarity Measures for Case-based Reasoning Systems", in *Proc. Fourth Intl. Conf. on Information Processing and Management of Uncertainty (IPMU-92) in Knowledge-Based Systems*, pp. 483-487, Palma, Spain, 1992.
- [116] P.P. Bonissone, W. Cheetham, D. Golibersuch, and P.S. Khedkar, "Automated Residential Property Valuation: An Accurate and Reliable Based on Soft Computing," in *Soft Computing in Financial Engineering*, R. Ribeiro, H. Zimmermann, R.R. Yager, and J. Kacprzyk, Eds., Heidelberg, Germany: Physica-Verlag (Springer-Verlag), 1998.
- [117] D. Bunn, "Combining Forecasts," *European Journal of Operations Research*, vol. 33, no. 3, pp. 223-229, 1988.
- [118] R. T Clemen, "Combining Forecasts: A review and Annotated Bibliography," *Int. Journal of Forecasting*, vol. 5, no. 4, pp. 559-584, 1989.
- [119] C.W.J Granger and R. Ramanathan, "Improved Methods of Combining Forecasts," *Journal of Forecasting*, 3, pp. 197-204, 1984.
- [120] E. Kacapyr, "Consensus Forecasts," *Economic Forecasting: The State of the Art*, M. E. Sharpe, Inc., 1996.
- [121] S. K. McNees, "Consensus Forecasts: Tyranny of the Majority," *New England Economic Review*, pp. 14-21, Nov-Dec 1987.