A hybrid-genetic algorithm for training a Sugeno-type Fuzzy Inference System with a mutable rule base

Christopher G. Coy
The University of Toledo

Follow this and additional works at: http://utdr.utoledo.edu/theses-dissertations

Recommended Citation
http://utdr.utoledo.edu/theses-dissertations/814

This Thesis is brought to you for free and open access by The University of Toledo Digital Repository. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of The University of Toledo Digital Repository. For more information, please see the repository's About page.
A Thesis

entitled

A Hybrid-Genetic Algorithm for Training a Sugeno-Type
Fuzzy Inference System with a Mutable Rule Base

by

Christopher G. Coy

Submitted to the Graduate Faculty as partial fulfillment of the
requirements for the Master of Science Degree in Engineering

______________________________________
Devinder Kaur, Ph.D., Committee Chair

______________________________________
Henry Ledgard, Ph.D., Committee Member

______________________________________
Ezzatollah Salari, Ph.D., Committee Member

______________________________________
Patricia R. Komuniecki, Ph.D., Dean

College of Graduate Studies

The University of Toledo

December 2010
An Abstract of

A Hybrid-Genetic Algorithm for Training a Sugeno-Type Fuzzy Inference System with a Mutable Rule Base

by

Christopher G. Coy

Submitted to the Graduate Faculty in partial fulfillment of the requirements for the Master of Science Degree in Engineering

The University of Toledo
December 2010

The accurate modeling of a time series using a Sugeno Fuzzy Inference System (FIS) requires an algorithm that can train the FIS to minimize the error of seen and unseen data points. Many researchers have used genetic algorithms to optimize the parameters of the FIS membership functions with a great deal of success. The local-search-augmented hybrid genetic algorithm presented here incorporates FIS rule base identification into the training process is shown to improve the accuracy of predicting future time series data points, by using the well-known Mackey-Glass time series and Box-Jenkins Glass Furnace data set as benchmarks. The main structural identification consists of optimizing the number of rules in the rule base. While the Mackey-Glass Time Series exercises the algorithm’s ability to model a highly complex aperiodic function, the Box-Jenkins Gas Furnace data set is used to show the algorithm’s ability to handle a smaller, noisier data set and provide a FIS that does not succumb to overfit.

The results support the claim that the mutable rule base presented here are an improvement over existing methods.
Acknowledgements

I would like to thank Dr. Devinder Kaur, my advisor, for her years of teaching, guidance, and patience. Without her help and encouragement I would surely spend another three years finishing this thesis. She has helped me learn about the academic industry in several ways, the greatest of which was presenting a paper we coauthored at the North American Fuzzy Information Processing Society’s annual conference.

I would also like to thank Becky DeKeyser, my loving girlfriend, who has allowed me to spend countless hours researching, experimenting, and writing – when I should have been fixing our home and improving our yard. She accompanied me on my journeys, including the conference in Toronto, and has been a big supporter of furthering my academic career.
Table of Contents

Abstract........................................................................................................................................ iii

Acknowledgments................................................................................................................ iv

Table of Contents ................................................................................................................ v

List of Tables ......................................................................................................................... xi

List of Figures ........................................................................................................................ xii

1 Introduction ......................................................................................................................1

1.1 Intro .........................................................................................................................1

1.1.1 Organization ......................................................................................................2

2 Fuzzy Inference Systems .............................................................................................4

2.1 Fuzzy Sets ...............................................................................................................4

2.1.1 Fuzzy Membership Functions .......................................................................5
2.2 Fuzzy Inference

2.2.1 Fuzzification Interface

2.2.2 Fuzzy Rules

2.2.3 Decision-Making Unit

2.2.4 Defuzzification

2.3 Fuzzy Models

2.3.1 Mamdani Model

2.3.2 TSK Model

2.3.3 Tsukamoto Model

2.4 Tuning Fuzzy Inference Systems

2.4.1 Adaptive Networks

2.5 ANFIS

3 Evolutionary Computation

3.1 An Introduction to Evolutionary Computation
3.2 Genetic Algorithms .......................................................................................... 17

3.2.1 Encoding in Genetic Algorithms ............................................................... 17

3.2.2 Selection in Genetic Algorithms .............................................................. 18

3.2.3 Crossover in Genetic Algorithms ............................................................ 20

3.2.4 Mutation in Genetic Algorithms .............................................................. 22

3.3 Real-Coded Genetic Algorithms .................................................................... 22

3.3.1 Crossover in RCGAs ............................................................................... 23

3.3.2 Mutation in RCSGAs ............................................................................... 24

3.4 Hybrid Genetic Algorithms ........................................................................... 24

4 Genetic Fuzzy Systems .................................................................................... 26

4.1 An Introduction to Genetic Fuzzy Systems .................................................. 26

4.2 Encoding GFRBS into Chromosome ............................................................ 27

5 Problem Definition .......................................................................................... 29

5.1 Mackey-Glass Equation .............................................................................. 29
5.2 Box-Jenkins Gas Furnace Data .................................................................32

6 Predicting the Mackey-Glass Time Series .....................................................35

6.1 Approach ..................................................................................................35

6.2 Encoding the Fuzzy Inference System ......................................................36

6.3 Initial Population ......................................................................................37

6.4 Fitness Function .......................................................................................39

6.4.1 Low Training Error .............................................................................39

6.4.2 Distinguishable Membership Functions ..............................................40

6.4.3 Complete Coverage of the Universe of Discourse ..............................41

6.4.4 Total Fitness Function ........................................................................42

6.5 Consequent Parameter Calculation ..........................................................43

6.6 Selection ..................................................................................................44

6.7 Crossover ................................................................................................46

6.8 Mutation ..................................................................................................49
7.10 Rule Base Pruning .................................................................67

7.11 Results on the Mackey-Glass Time Series..........................68

7.12 Results on the Box-Jenkins Gas Furnace Data Set...............72

7.13 Conclusions ......................................................................75

8 Recommendations for Future Work .........................................77

8.1 Improved Overfit Detection .................................................77

8.2 Consequent Parameter Calculation ........................................77

8.3 Additional Antecedent Membership Functions ......................78

8.4 Support for Mamdani-Style FIS .........................................78

8.5 Use of Fuzzy Clustering ....................................................79

References ..............................................................................80
List of Tables

Table 6.1 Algorithm Parameters .................................................................53

Table 6.2 RMSE for Mackey-Glass ...............................................................54

Table 6.3 Comparative Results .................................................................56

Table 7.1 Results for Modifiable Rule Base ...............................................69

Table 7.2 FIS Rule Mappings .................................................................71

Table 7.3 Box-Jenkins Trained with ANFIS ..............................................72

Table 7.4 Box-Jenkins Parameters .............................................................73

Table 7.5 Box-Jenkins Rule Mappings .....................................................75
List of Figures

Figure 2-1 Temperature Membership Functions ...............................................................4

Figure 2-2 Fuzzy Inference System ..................................................................................6

Figure 2-3 Adaptive Network ........................................................................................ 12

Figure 3-1 Evolutionary Algorithm Flow ....................................................................... 16

Figure 3-2 One-Point Crossover .................................................................................... 21

Figure 3-3 Two-Point Crossover .................................................................................... 22

Figure 4-1 GFRBS Encoding ......................................................................................... 27

Figure 5-1 Mackey-Glass Time Series ........................................................................... 31

Figure 5-2 Box-Jenkins Gas Furnace Input .................................................................... 33

Figure 5-3 Box-Jenkins Gas Furnace Output .................................................................. 33

Figure 6-1 FIS Encoding ................................................................................................. 36
Figure 6-2 Relative Number of Membership Functions .................................................. 38

Figure 6-3 Distinguishability of Membership Functions ............................................... 40

Figure 6-4 Overlap Penalty ............................................................................................ 41

Figure 6-5 Selection and Crossover ............................................................................... 45

Figure 6-6 Mutation and Elitism .................................................................................... 46

Figure 6-7 Single-Point Crossover ............................................................................... 47

Figure 6-8 N-Point Crossover ....................................................................................... 48

Figure 6-9 Mutation ....................................................................................................... 50

Figure 6-10 Fitness of Decaying Mutation ..................................................................... 55

Figure 6-11 Fitness of Auto-Tuning Mutation ............................................................... 56

Figure 6-12 Training Error Using 3 MFs per Input ......................................................... 57

Figure 7-1 FIS Structure ............................................................................................... 60

Figure 7-2 Modifiable Rule Base Algorithm .................................................................. 62

Figure 7-3 RMS Error Plot for Modifiable Rule Base .................................................... 70
Figure 7-4 Error Plot for Box-Jenkins using Mutable Rule Base ........................................74
Chapter 1

1 Introduction

1.1 Intro

Fuzzy inference systems (FIS) are used in a number of fields for the purposes of function modeling, control, and classification, among others. When modeling functions, an FIS can approximate a highly nonlinear system via a group of rules. Originally, experts tuned the FIS by manually adjusting the parameters, which was an expensive effort. In his 1993 paper entitled, “ANFIS: Adaptive-Network-Based Fuzzy Inference System,” Roger Jang introduces a novel approach for automated FIS tuning (Jang, 1993). ANFIS is similar to a neural network, wherein the antecedent parameters in the FIS are trained via gradient descent, and the consequent parameters are trained via least-squares estimation. The 5-layer architecture of ANFIS consists of a membership function layer, firing strength layer, normalized firing strength layer, consequent parameter layer, and a summation or output layer. The objective function for ANFIS is simply the training error, which is to be minimized. Jang uses several example applications as benchmarks for ANFIS, to demonstrate its ability to represent a nonlinear function and to predict future data points of a time series. The chaotic time series was the Mackey-Glass
differential delay equation defined as shown in Equation 1.1, and has been studied by several researchers.

\[
\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \quad \text{(Eq. 1.1)}
\]

Jang follows the parameters used by prior researchers, with initial conditions: \( x(0) = 1.2, \tau = 17,118 \leq t \leq 1117 \). Each data example into ANFIS consisted of 4 inputs and 1 output following the convention in Equation 1.2 below.

\[
Data(t) = \{(x(t - (D - 1)\Delta), ..., x(t - \Delta), x(t) \xrightarrow{\text{predicts}} x(t + P)}\} \quad \text{(Eq. 1.4)}
\]

Here \( \Delta = P = 6 \) represents the input time interval, and prediction time offset, and \( D = 4 \) is the number of inputs. To clarify, the data examples used follow equation 1.3.

\[
Data(t) = \{(x(t - 18), x(t - 12), x(t - 6), x(t) \xrightarrow{\text{predicts}} x(t + 6)}\} \quad \text{(Eq. 1.3)}
\]

### 1.1.1 Organization

The thesis is organized as follows. Chapter 2 describes Fuzzy Sets and Fuzzy Inference Systems, as well as ANFIS. Chapter 3 focuses on genetic algorithms and evolutionary computation. Chapter 4 combines the prior two, discussing genetic fuzzy systems, and how to train an FIS using evolutionary computation, as well as the algorithmic decisions made in this work. Finally, in Chapter 5 the results are shown and conclusions are drawn, with some segues to additional areas of study.
The purpose of this work is to demonstrate an improvement upon the prior works for the benchmarks common to the community at large, representative of real-world problems. It will be shown that, through careful design, the goals of increased generalization, reduced training and validation error, and reduced rule base can be achieved. The keystone of this algorithm is its flexibility, allowing for an evolving FIS architecture in addition to identifying optimal membership function parameters.
Chapter 2

2 Fuzzy Inference Systems

2.1 Fuzzy Sets

Fuzzy logic, introduced by Lofti Zadeh in 1965, extends upon classical logic by generalizing the notions of “true” and “false” to accommodate reasoning under uncertainty (Zadeh, 1965).

Figure 2-1 Temperature Membership Functions
2.1.1 Fuzzy Membership Functions

Each variable has a degree of participation in every fuzzy set, based quantitatively on a membership function. An example of a fuzzy set is the set of temperatures that are considered to be hot. A comparative crisp, or classical, set could be temperatures greater than 80°F. The membership function of the fuzzy set expresses a variable’s degree to which it belongs to the set, ranging from 0 to 1, inclusive. Most membership functions are defined by a few number of parameters, and therefore common function are often triangular, trapezoidal, Gaussian, or bell-shaped. Figure 2-1 illustrates some membership functions for fuzzy sets dealing with temperature. Here, a temperature of 50°F would have a membership of 0.2, 0.4, and 0 in the sets cold, mild, and hot, respectively.

Many fuzzy set designers try to follow certain guidelines when defining membership functions: 1) a MF should only overlap with the closest neighboring MFs, and no others, and for any possible input value on the universe of discourse, 2) the sum of the membership degree in all fuzzy sets should be 1, and 3) MFs should be symmetric (for system stability). These guidelines are optional, and depending on the problem at hand, can be waived. (Yen & Langari, 1999) As seen in Figure 2-1, the membership functions depicted do not meet the second criteria.

2.2 Fuzzy Inference
A Fuzzy Inference System (FIS), as seen in Figure 2-2, uses membership functions to “fuzzify” input data, and then applies a set of rules (rule base) to the fuzzy data. An FIS can be used for system control, function approximation, time series prediction, and other applications.

![Figure 2-2 Fuzzy Inference System](image)

2.2.1 Fuzzification Interface

The fuzzification interface performs a fuzzy matching step, calculating the membership degree the input has to each fuzzy set associated with the input. The fuzzy membership degrees are passed along to the decision-making unit, which uses a set of fuzzy rules to compute a fuzzy output. Finally, the defuzzification interface converts the fuzzy conclusion into a crisp one. (Yen & Langari, 1999)

2.2.2 Fuzzy Rules
A fuzzy rule consists of an antecedent and a consequent, like other hypothetical propositions. The antecedent here contains one or more conditions connected by conjunctions or disjunctions. For example, “If the temperature is hot and the humidity is low, then set the sprinkler output to high.” In this case, “temperature” and “humidity” represent input variables, while “hot” and “low” represent fuzzy sets, defined by fuzzy membership functions. The consequent contains “high” which is another fuzzy set belonging to the sprinkler output variable. The antecedent is evaluated in the fuzzification step, by computing the membership degree each variable has in their associated fuzzy sets, and evaluating the connectives as defined by the system. Often the connectives are AND operations (conjunctions), which equate to min operators or possibly multiplicative products. Likewise, OR connectives (disjunctions) are typically max operators or additive summations. The evaluated antecedent result is known as the firing strength of the rule or matching degree (Yen & Langari, 1999).

### 2.2.3 Decision-Making Unit

The decision-making unit performs the inference operations on the fuzzy data to obtain the conclusion of each rule. The conclusions of these rules are combined using a fuzzy disjunction operator (usually max or sum) (Yen & Langari, 1999).

### 2.2.4 Defuzzification

Defuzzification is an optional step performed when a crisp output is required. Common techniques include the Mean of Maximum (MOM) method and the Center of Area (COA) or centroid method. MOM calculates the average of the highest-valued
conclusions for each rule. This has the limitation that it does not consider the shape or
distribution of the aggregated conclusions. COA is used more often because it does not
suffer from that limitation, albeit at the expense of a higher computational cost.

\[
COA(A) = \frac{\sum x \mu_A(x) \cdot x}{\sum x \mu_A(x)}
\] (Eq. 2.1)

As shown in Equation 2.1, COA calculates the weighted average of the rule firing
strengths (Yen & Langari, 1999).

A third method, sometimes used in neuro-fuzzy models, is the Height Method, wherein
the consequent membership functions have crisp outputs and the defuzzification step
consists of a weighted average of the rule strengths. This is shown below in Equation
2.2.

\[
y = \frac{\sum_{i=1}^{M} w_i c_i}{\sum_{i=1}^{M} w_i}
\] (Eq. 2.2)

\(w_i\) is the firing strength of a rule and \(c_i\) is the centroid of the associated consequent MF.

It is simple and fast, but is considered an approximation to COA (Yen & Langari, 1999).

### 2.3 Fuzzy Models

Traditionally, the designer of the FIS would define the membership functions by hand,
based on expert knowledge, as well as maintaining constraints that map the entire range
of inputs to appropriate fuzzy sets. The membership functions (MFs), stored in the
database, are usually represented with few parameters, such as functions that are triangular, trapezoidal, or Gaussian in nature, or a bell curve. Typically, the rule set is a collection of all the possible membership function combinations, with one membership function per input; however one could design a FIS with fewer rules, if some of these combinations were impractical. The standard FIS models differ slightly in their inference schemes, where some combine rule results with superimposition, and some use a weighted sum. The models discussed here are Mamdani, Takagi-Sugeno-Kang (TSK), and Kosko’ additive model (SAM).

### 2.3.1 Mamdani Model

A Mamdani model is a commonly used FIS wherein the rules are in the form of Equation 2.3.

\[
R_i: \text{IF } x_1 \text{ is } A_{i1} \text{ and } \ldots \text{ and } x_n \text{ is } A_{in} \text{ THEN } y \text{ is } C_i. \quad (\text{Eq. 2.3})
\]

Where \( x_j \) represents the input variables, \( y \) the output, \( A_{ij} \) the antecedent fuzzy set for rule \( i \) and input \( j \), and \( C_i \) the consequent fuzzy set for rule \( i \). The firing strength of the rule here is the minimum of the evaluations of the membership functions for a particular set of inputs. The membership functions are defined by Equation 2.4 below.

\[
\mu_R(x, y) = I(\mu_{A}(x), \mu_{C}(y)) \text{ and } \mu_{A}(x_0) = T(\mu_{A1}(x_1), \ldots, \mu_{An}(x_n)) \quad (\text{Eq. 2.4})
\]

In Equation 2.4, \( T \) is a triangular norm (t-norm) fuzzy conjunctive operator, usually the minimum function. The defuzzified output is often the maximum of the membership
functions. The Mamdani model is easy to understand, and can be human interpretable (Yen & Langari, 1999). However, the simplistic rule structure can make it difficult to model highly complex problems in a concise manner.

### 2.3.2 TSK Model

Takagi, Sugeno, and Kang developed a more compact FIS that is much more expressive, at the expense of human interpretability. The antecedents and membership functions are similar to that of the Mamdani FIS, but the consequent is a function instead of a fuzzy set. Typically this function is a first order polynomial. As such, a Mamdani FIS could be viewed as a 0-th order TSK FIS. Shown below in Equation 2.5 is the TSK rule structure.

\[
R_i: \text{IF } x_1 \text{ is } A_{i1} \text{ and } … \text{ and } x_n \text{ is } A_{in} \text{ THEN } y = C_{i1}x_1 + … + C_{in}x_n + C_{in+1} \quad \text{(Eq. 2.5)}
\]

The main advantage over the Mamdani model is the reduced number of rules. Another difference between the two types of FIS is that the TSK system often uses a weighted average of the rule outputs, instead of the max operator. The fact that the rule output (depicted as y above) is crisp also reduces any potentially computationally expensive defuzzification process. The TSK is also commonly referred to as the Sugeno model (Yen & Langari, 1999).

### 2.3.3 Tsukamoto Model
A third, less commonly used model is the Tsukamoto model. It is similar to the TSK model, but the rule consequent function is monotonically increasing or decreasing, and typically between 0 and 1, inclusive, similar to fuzzy membership functions.

### 2.4 Tuning Fuzzy Inference Systems

Experts in the field would manually adjust the parameters of the FIS in order to make it appropriate for use in a control system or otherwise. This was possible using Mamdani models, but often would require a great number of rules. Increasingly complex systems needed to be optimized via automated means, and as such, algorithms were developed to do so. One way to automate this is via machine learning, and more specifically, by using an adaptive network.

#### 2.4.1 Adaptive Networks
An adaptive network is a multilayer feed-forward neural network in which each node performs a function on incoming signals and on a set of parameters.

The links between the node layers indicate only the flow direction of signals, and not weights as in other neural network architectures. The parameters of the nodes are trained based on reducing the error measure of the training data, using gradient descent. The gradient descent method iteratively multiplies a learning rate system parameter to the parameters of each node based on the rate of change of the error metric. The learning rate can be adjusted automatically to aid convergence (Jang, 1993).
2.5 ANFIS

ANFIS, proposed by Jang in 1993, is an Adaptive-Network-Based Fuzzy Inference System. It uses a network like the adaptive network described above to represent and optimize a Mamdani or TSK FIS. In addition to using strictly gradient descent for parameter optimization, the consequent parameter nodes can be calculated via least squares estimation, thus reducing the dimensionality of the search problem. Keeping the antecedent parameters fixed temporarily, the consequent parameters can be calculated by placing the evaluated antecedents, parameters to train, and expected output (from the training data) in a matrix equation in the form of $AX = B$, respectively. Jang uses a sequential least-squares estimate to compute $X^*$, or the covariance matrix. (Jang, 1993)

Specifically, the ANFIS architecture consists of 5 layers, as illustrated above in Figure 2-3 Adaptive Network Architecture diagram. The first layer contains the membership functions, $\mu_{A_i}(x)$, which is typically a bell or Gaussian curve, with a maximum of one and minimum of zero. Layer two computes the firing strength of each rule, as shown in Equation 2.6.

\[
    w_i = \mu_{A_i}(x) \ast \mu_{B_i}(x) \quad \text{(Eq. 2.6)}
\]
The * in Equation 2.6 can represent any T-norm operator, though it is typically multiplication. The third layer normalizes the firing strength of each rule by dividing it by the sum of all the rules’ firing strengths (outputs from the second layer).

The nodes in the fourth layer in ANFIS architecture contain the consequent parameters, which are evaluated against the data, summed, and multiplied by the normalized firing strength of the rule. The final layer has but one node, which aggregates the output of all the rules (layer 4 outputs). More complex systems have one node per output (Jang, 1993)
3 Evolutionary Computation

3.1 An Introduction to Evolutionary Computation

Evolutionary algorithms (EAs) are search and optimization algorithms that imitate natural evolution. Different forms of evolutionary computation are genetic algorithms, evolutionary programming, genetic programming, and grammatical evolution, among others. All EAs focus on having a population of candidate solutions, combination of those solutions, and mutation of the solutions to generate new solutions. In order to achieve convergence, the EAs must also rank and select solutions for the next population to increase the fitness of the population. The figure below illustrates the flow of a typical EA.
As mentioned above, an EA has a population of candidate solutions, which are known as chromosomes. These chromosomes consist of genes, as in natural biology, however a gene in an EA context contains a specific parameter to be optimized. Also inspired by natural biology is the process for creating new chromosomes: selection, crossover, and mutation. Selection, discussed in section 3.2.2, ranks the chromosomes based on their ability to solve the problem (usually minimizing training error) and then chooses two “parents” with a probability related to their rank in the population. Likewise, poor solutions are unlikely to be chosen during the selection process, so their genetic material is eventually lost. The ability to solve the problem is known as fitness, like in Charles Darwin’s “Survival of the Fittest” theory, and is measured by the fitness function.
Crossover, explained in detail in section 3.2.3, is the recombination process that takes some of the genes from each selected parent and forms a complete chromosome fit to be a member of the subsequent generation (iteration). The goal of this operation is to find a chromosome that contains the best genes from each parent.

Mutation, discussed in section 3.2.4, also modifies the new chromosome, but typically is a one-in, one-out operation where a gene is randomly replaced or modified without the influence of a parent’s genetic material. The goal of mutation is to maintain diversity by creating new genetic material, which may not exist in the rest of the population. It helps in avoiding local minima during the global search process.

3.2 Genetic Algorithms

Genetic Algorithms (GAs) are global search heuristics that evolve solutions to problems. Invented by John Holland in 1975, they have been widely used and researched since, due to their usefulness in traversing complex search spaces, such as NP-hard problems (Holland, 1975). In order to use a GA, the designer must figure out how to encode the solutions into chromosomes, how to create a suitable random initial population, how to evaluate the candidate solutions via a fitness function, how to employ crossover and mutation, and the GA parameters, including population size, number of generations, and probability of crossover and mutation.

3.2.1 Encoding in Genetic Algorithms
Many GAs encode the parameters to optimize as binary strings. This simplifies many things, especially the genetic operators of crossover and mutation. For example, one can decode a variable from a subset (n bits) of a binary string $s_0, \ldots, s_N$ with bits $s_i \{0,1\}$ to represent a parameter, $y$, as shown in equation 3.1.

$$y = y_{\text{min}} + \frac{y_{\text{max}} - y_{\text{min}}}{2^n - 1} \times \sum_{i=0}^{n-1} s_i \times 2^i$$

(Eq. 3.1)

The more bits (larger n) that are used represent increased resolution of the parameter. This simple scheme can cause problems in mutation, since two very different binary strings may represent two similar integers, such as 1000 and 0111 when representing 8 and 7, respectively. To avoid this, many GAs use Gray code, which guarantees adjacent integers are represented by binary strings that only differ in a single bit (Cordon, Herrera, Hoffmann, & Magdalena, 2001).

For a simple binary-encoded GA, the initial population can be easily generated by populating each binary string with random 1’s and 0’s. More complex problems may impose additional restraints on the chromosomes, and these restraints must be enforced at this time as well as during the genetic operators.

### 3.2.2 Selection in Genetic Algorithms

To assess the fitness of the chromosome, typically the chromosome must be decoded into its constituent variables, which are evaluated using the fitness function. Often, this will be an error metric that is to be minimized, but it is problem-specific and can be a different
metric to either minimize or maximize. This must be designed in conjunction with the selection operation in order to appropriately select parents for the next generation.

As mentioned previously, selection chooses parent chromosomes for the subsequent generation. With the goal of converging upon an optimal solution, the selection mechanism should select stronger, fitter solutions more often than the less-fitting ones. Based on the fitness function, each chromosome has a fitness value, which is used to rank the chromosomes in the population. One simple scheme for selection is the roulette wheel sampling, where each chromosome \( C_i \) in \( P \) has a selection probability as in equation 3.2.

\[
P_s(C_i) = \frac{f(C_i)}{\sum_{i=1}^{N} f(C_i)} \tag{Eq. 3.2}
\]

The roulette wheel sampling makes the chance for reproduction directly proportional to its fitness. This has limitations, one being the fitness values must all be positive. Another significant drawback is that a chromosome with a very high relative fitness value can dominate the population, and be chosen most of the time, causing premature convergence. Fitness scaling methods avoid this by adjusting the fitness values based on the distribution of fitness values for the entire population. For example, linear scaling creates new fitness values using equation 3.3.

\[
\hat{f}(C_i) = c_0 * f(C_i) + c_1 \tag{Eq. 3.3}
\]
In Equation 3.3 $c_0$ and $c_1$ are adjusted such that the expected number of times a chromosome is chosen becomes $\alpha \approx 1.5 \ldots 3.0$ and an average chromosome with fitness $\bar{f}$ is chosen once. As such, one can choose $c_0 = \alpha - 1$ and $c_1 = f_{max} - \bar{f}$. Similarly, sigma truncation scales the fitness values based on the standard deviation $\sigma$ of the fitness of the population using equation 3.4 (Cordon, Herrera, Hoffmann, & Magdalena, 2001, pp. 61-62).

\[
\hat{f}(C_i) = \frac{f(C_i) - f - \sigma}{\sigma}
\]  
(Eq. 3.4)

Ranking also avoids premature convergence by sorting the population based on the fitness values, and then performing selection based on the chromosome’s rank.

Elitism is another selection variant in which the best chromosome is maintained as is in the subsequent generation. It insures that the optimal solution is never lost via the genetic operators.

### 3.2.3 Crossover in Genetic Algorithms

Crossover is the most important operation in a genetic algorithm, because it recombines the useful genetic information of the parent chromosomes to create the next generation of solutions. This crossover operation is typically performed with a probability of occurrence of 0.6 to 0.95. The simplest form of crossover is the one-point crossover, in which a position in the binary string is chosen randomly, and the parents swap segments there to form the two new chromosomes.
The main downside of one-point crossover is that the genes on the ends of the chromosome are exchanged more often than those in the middle. Two-point crossover mitigates this, by selecting two locations and exchanging the genes in between them. Uniform crossover is an extension of this, where each gene is independently chosen from one of the parents.
3.2.4 Mutation in Genetic Algorithms

The goal of mutation is to create new genetic material in the genetic algorithm, exploring new values for genes previously not in the population. The basic method is randomly inverting a bit of the binary string, with a probability of occurrence chosen by the designer, but normally in the interval [0.001, 0.02]. This can be abstracted to the chromosome level, by selecting a particular bit or substring of bits to invert.

3.3 Real-Coded Genetic Algorithms

Many times it is easier to use a vector of real numbers as chromosomes instead of binary strings, due to increased interpretability and less work encoding and decoding. Real-
coded genetic algorithms (RCGAs) are often used for optimizing continuous domain problems (instead of binary-coded GAs for combinatorial problems).

### 3.3.1 Crossover in RCGAs

Simple crossover, with one or more crossover points, can still be used to exchange genetic material between two parent chromosomes. Similarly, discrete crossover chooses genes randomly from one of two parents like uniform crossover in binary-coded GAs. New crossover schemes were developed to mix the genetic material while searching the continuous domain of each gene, or search parameter. Flat crossover creates an offspring \( C' \) from parents \( C_1 \) and \( C_2 \), described below. Equation 3.5 illustrates the chromosome representation of the parents (or any chromosome), and Equation 3.6 is the flat crossover function, where the value \( \lambda_i \) dictates the weight each parent contributes to the new children chromosomes.

\[
C_1 = (x_1^1, \ldots, x_n^1) \quad \text{and} \quad C_2 = (x_1^2, \ldots, x_n^2) \quad \text{(Eq. 3.5)}
\]

\[
x_i' = \lambda_i * x_i^1 + (1 - \lambda_i) * x_i^2; \quad i \in \{1, \ldots, n\} \quad \text{(Eq. 3.6)}
\]

Here, \( \lambda_i \) is a uniformly distributed random number in \([0, 1]\).

BLX-\( \alpha \) crossover is a modified crossover function, which increases the search interval to go beyond the values of the parents’ genes (Cordon, Herrera, Hoffmann, & Magdalena, 2001, p. 65). In BLX- \( \alpha \), the range is determined by \( \alpha \), which is chosen by the designer.
3.3.2 Mutation in RCSGAs

Often in a real-coded GA, convergence is increased with the use of a decreasing mutation interval. One way to perform this is through a non-uniform mutation, where $t$ is the current generation and $t_{\text{max}}$ is the maximum number of generations, and

\[
x_i' = \begin{cases} 
  x_i + \Delta(t, b_i - x_i), & \text{if } \tau = 0 \\
  x_i - \Delta(t, x_i - a_i), & \text{if } \tau = 1
\end{cases} \quad | P(\tau = 0) = P(\tau = 1) = 0.5 \quad (\text{Eq. 3.8})
\]

\[
\Delta(t, x) = x \times (1 - \lambda \left(1 - \frac{t}{t_{\text{max}}}\right)^b) \quad | \lambda = \text{random number in } [0, 1] \quad (\text{Eq. 3.9})
\]

3.4 Hybrid Genetic Algorithms

Genetic algorithms are used for their ability to fully explore the search space and eventually converge upon the global optimum value, though it may take several generations. In contrast, local search methods converge more rapidly, but they may only find the local optimum value. However, researchers have been studying the use of different local search heuristics to augment the global search process. The hybrid genetic algorithm is simply a GA combined with local search elements (Russo, 2000).

Common local search techniques used in hybrid GAs are nearest-neighbor, gradient descent, and steepest descent, among others. Each has a set of tradeoffs, and are typically chosen based on the underlying GA. For example, a real-coded GA with a continuous
(differentiable) search space may use gradient descent, whereas a discrete combinatory GA may use a nearest-neighbor approach. A major factor to consider in selecting the appropriate local search method is the tradeoff between computation time and the algorithm’s ability to find a reasonable local optimum. The local optimum can be difficult to locate when several inputs are to be considered, as the search space can be quite rigid. (Cordon, Herrera, Hoffmann, & Magdalena, 2001)
4 Genetic Fuzzy Systems

4.1 An Introduction to Genetic Fuzzy Systems

A Genetic Fuzzy System (GFS) is an FS that learns via a genetic process to tune different components of the FS. A GFS can be a genetic fuzzy rule-based system (GFRBS), genetic fuzzy clustering system, genetic fuzzy neural network, or possibly a genetic fuzzy decision tree. The GFRBS is the focus of the work here, and as such, is discussed further.

GFRBSs reduce the amount of knowledge and effort on the part of an expert in designing the inference system. The evolutionary computation automates the design steps of the FRBS by learning the rules, and parameters thereof. This is an optimization problem of a vast search space, potentially, and therefore a GA is well-suited to find a global optimum.

The GFRBS designer must identify which parameters to optimize, for there is a tradeoff between finding the best FIS and the complexity and length of the search. Some of the parameters that could be optimized are scaling function bounds and dilation factors, membership functions, and rule mappings. Due to the often continuous nature of these
parameters, most designers choose to implement a real-coded GA or develop special
genetic operators to work on the FIS (Cordon, Herrera, Hoffmann, & Magdalena, 2001).

4.2 Encoding GFRBS into Chromosome

The most common way to train a Genetic Fuzzy System (GFS) is to simply serialize the
membership function parameters into a real-coded string, and use the real-coded GA
techniques described in section 3.3.

Figure 4-1 GFRBS Encoding

Shown above in Figure 4-1 is an example of creating a chromosome from an FIS with 3
inputs. Each input has two membership functions (MFs) associated with it, which are
both Gaussian curves. The Gaussian membership functions are represented by two
parameters: a center value and a width value. Gaussian membership functions are
commonly used because they are continuous, differentiable, and easy to represent.
Assuming the rule base remains constant, as it does in ANFIS (described in section 2.5), and there is one rule for each combination of membership functions when choosing one per input, the chromosome is simply this vector of MF parameters.
Chapter 5

5 Problem Definition

5.1 Mackey-Glass Equation

Michael Mackey and Leon Glass describe a first-order nonlinear differential-delay equation modeling arterial CO$_2$ control system in their 1977 paper (Mackey & Glass, 1977).

\[
\dot{x}(t) = \lambda - \frac{\alpha V_m x^n x^*(t-\tau)}{\theta^n + x^n(t-\tau)}
\]  
(Eq. 5.1)

Here, \(\lambda\) is the CO$_2$ production rate, \(\tau\) is the time between oxygenation of blood in the lungs and stimulation of chemoreceptors in the brainstem, and \(\alpha\) is a constant. \(x(t)\) represents the concentration of CO$_2$ in the blood at time \(t\) when the blood is produced.

In some patients with abnormal pathologies, such as leukemia, the delay time \(\tau\) may become quite large, and the concentration of CO$_2$ in the blood will oscillate, becoming chaotic when \(\tau \geq 17\). This problem is important because it can help doctors and researchers to develop therapies for disease by adjusting the control parameters to a normal range.
Due to its chaotic nature and similarity to other real-world problems, the Mackey-Glass equation has been used extensively as a benchmark in Neural Network training, and other machine learning algorithms, beginning with Lapedes and Farber in 1987 (Lapedes & Farber, 1987). The form of the equation normally used is as follows in equation 5.2.

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \quad \text{(Eq. 5.2)}$$

To generate the time series, the following initial conditions are set: $x(0) = 1.2$, $\tau = 17, 118 \leq t \leq 1117$. This ensures a large enough standard data set to train, test, and validate the learning algorithm. Also, the $\tau = 17$ condition makes the time series chaotic, or aperiodic. This time series is illustrated below, for $0 \leq t \leq 1200$. 
From the time series data, training data examples are created by sampling the series at a uniform time interval, denoted by $\Delta$ in the equation below. There are $D$ inputs, which are used to predict the output – the value of the time series at a time $P$ in the future.

$$\text{Data}(t) = \{x(t - (D - 1)\Delta), ..., x(t - \Delta), x(t) \xrightarrow{\text{predicts}} x(t + P)\} \text{ (Eq. 5.3)}$$

To replicate the experiments of prior researchers, $\Delta = P = 6$ is substituted into Equation 5.3 for the input time interval and prediction time offset, and $D = 4$ for the number of inputs. Equation 5.4, a specific instance of Equation 5.3 with the values specified above, is used to generate the data samples for a given time value, $t$, where $118 \leq t \leq 1117$. 
\[ \text{Data}(t) = \{(x(t - 18), x(t - 12), x(t - 6), x(t))^{\text{predicts}} \rightarrow x(t + 6)\} \text{ (Eq. 5.4)} \]

5.2 Box-Jenkins Gas Furnace Data

The other data set used to validate the algorithm is the Box-Jenkins benchmark, containing 296 pairs of input-output samples representing the methane flow rate and the CO\(_2\) emissions of a gas furnace (Box & Jenkins, 1970). This is a smaller data set, and thus challenges the learning process. Also, this data is measured, not computed, and therefore is noisier than the Mackey-Glass equation. As such, the Box-Jenkins benchmark will test the generalization ability of the algorithm. The input methane flow rate is illustrated in Figure 5-2, and shall be referred to as \(u(t)\). Figure 5-3 shows the percentage of CO\(_2\) output from the furnace, later referred to as \(y(t)\). Note that the time samples are at 9 second intervals.
Figure 5-2 Box-Jenkins Gas Furnace Input

Figure 5-3 Box-Jenkins Gas Furnace Output
The training and validation sets used to solve this problem vary, due to perceived predictability. The goal is similar to the Mackey-Glass time series prediction problem, but here the current CO$_2$ output, $y(t)$, is to be predicted from the input methane flow rate from 3 time samples ago, $u(t - 4)$, the current methane flow rate, the CO$_2$ output from 4 time samples ago, as well as the last measured CO$_2$ output value, $y(t - 1)$.

$$\mathbf{Data}_{trn}(t) = \left[ u(t - 3), u(t), y(t - 4), y(t - 1) \xrightarrow{\text{predicts}} y(t) \right] \quad (\text{Eq. 5.5})$$

In Equation 5.5 above, $u(t)$ represents the methane flow rate at a time sample $t$, and $y(t)$ represents the percentage of CO$_2$ output from the furnace.
Chapter 6

6 Predicting the Mackey-Glass Time Series

6.1 Approach

The Mackey-Glass Problem, explained in Chapter 5 of this thesis, is a useful benchmark to exercise the ability of an inference system to predict future values of a chaotic time series. In order to best predict the future values of \( x(t) \), the data are split up into training and test sets. Each data example follows the form shown in Equation 6.1 (also 5.4).

\[
Data(t) = \{(x(t - 18), x(t - 12), x(t - 6), x(t) \rightarrow x(t + 6)\} \quad (\text{Eq. 6.1})
\]

1000 data examples are generated from the time series equation in Equation 5.2, where the first 500 are used as training examples, and the second 500 are used as test examples, to be consistent with prior studies.

A Takagi-Sugeno-Kang type fuzzy inference system, as described in section 2.3.2 is used to predict the output. To create a suitable FIS, a hybrid genetic algorithm is used, as discussed in Chapters 3 and 4, wherein the population consists of fuzzy inference systems encoded into chromosomes, and custom genetic operators evolve the population in order
to train the FIS parameters. This chapter of the thesis thoroughly explains the design of this hybrid GA.

6.2 Encoding the Fuzzy Inference System

To train the parameters of the fuzzy inference system (FIS), the membership function parameters are serialized into a real-coded string, which makes up the chromosome.

![FIS Encoding Diagram](image)

Figure 6-1 is a graphical depiction of the chromosome representation of the FIS used to fit the training data of the Mackey-Glass time series problem. The four inputs each have two associated Gaussian membership functions. The rule base remains constant, wherein there are 16 rules, one for each combination of membership functions. This is important to note, because selecting too many membership functions increases the curse of dimensionality, wherein computation time grows exponentially as the number of membership functions increases.
6.3 Initial Population

The initial population consists of several fuzzy inference systems, created by grid partitioning of the training data, with the same properties as ANFIS, except with Gaussian membership functions (Jang, 1993). The maximum number of membership functions is provided by the user, and here is set to 2 per input, and likewise the minimum number is set to 1. The actual number of membership functions used for each input is determined by Equation 6.2 below.

\[
\text{NumMF} = \max(\min(\text{floor}(\max MF \times \text{rand} + \min MF), \max MF), \min MF) \quad (\text{Eq. 6.2})
\]

Above, rand is a uniformly distributed random variable from 0 to 1, inclusive. In practice, this produces a likelihood of the max value chosen twice that of the other values, as per the graph below.
Once the number of membership functions is decided, an ANFIS-compatible TSK FIS structure is created, using grid partitioning to locate the centers of the Gaussian membership functions based on the range of the input training data. Next, each input variable is evaluated to determine the bounds, and the center and sigma values of the Gaussian membership function are set based on random values within the appropriate ranges and distributions. The center point of each MF is chosen with the goal of distributing the MFs throughout the whole range, and thus the range is partitioned and then each center value is chosen randomly within its respective partition. The sigma
value, responsible for the width of the Gaussian function, is multiplied by \((0.2 + 0.8 \times \text{rand})\). The goal with this is to provide a sigma that will not make the MF too general or wide, nor too narrow and thus not representing any of the data, yet random in nature to differ from other MFs in other chromosomes. Given enough chromosomes in the population, there should be enough genetic material to identify optimal FIS parameters.

### 6.4 Fitness Function

The fitness function, or objective function, measures the ability of a chromosome to meet the desired criteria, and is used in global and local search methods. The one designed here attempts to optimize three goals: a low training error, a distinguishable set of membership functions, and complete coverage of the universe of discourse. To balance the three goals, the error sources are quantified, normalized, and summed, to produce a fitness function that best represents the desirability of the chromosome (or FIS).

#### 6.4.1 Low Training Error

The first goal is evaluated by first decoding the chromosome back into a FIS, and then computing the mean-squared error (MSE) of the system on the training data. The error is computed by subtracting the FIS output from the expected output for each training example, and then squaring the difference, and averaging the squares, as shown in
Equation 6.3. Similar metrics used are normalized mean-squared error (NMSE), and root mean-squared error (RMSE).

\[
MSE = \frac{1}{N} \left( \sum_{i=0}^{N} (y_{trn}(i) - y(i))^2 \right)
\]  
(Eq. 6.3)

### 6.4.2 Distinguishable Membership Functions

The second goal, finding a set of distinguishable membership functions, is optimized by analyzing the membership functions for each input and calculating the amount the MFs overlap each other, similar to other works (Kim, Kim, & Lee, 2006). This is important, because indistinguishable MFs diminish the representative power of the fuzzy model, and therefore can worsen the performance of the FIS throughout the evolutionary process.

![Figure 6-3 Distinguishability of Membership Functions](image)

As seen in Figure 6-3, the membership functions in (a) are overlapping quite a bit, whereas in (b) they are distinct, and therefore interpretable. In (b), however, there is a
low total membership degree at some values, e.g. -0.5. The third goal helps to prevent that by ensuring a certain level of coverage for the entire universe of discourse.

### 6.4.3 Complete Coverage of the Universe of Discourse

To ensure the membership functions do not overlap too much (indistinguishable) and that they fully cover the universe of discourse for their respective input variable, two penalty functions are computed with thresholds set by the designer.

![Figure 6-4 Overlap Penalty](image)

In Figure 6-4 above (Kim, Kim, & Lee, 2006), the overlap penalty threshold is denoted as $\xi_D$. For an input, $i$, there are $M-1$ overlapping regions, where $M$ is the number of membership functions. The length of the $i^{th}$ universe of discourse, $\chi_i$, is defined by the...
difference of the input’s min and max values found in the training data. The first penalty
function calculates the percentage of the universe of discourse covered by more than one
MF above $\xi_D$. This is done by calculating the intersection points ($Int$) of the MF with the
horizontal line at $\xi_D$, and then evaluating them as follows.

$$Int = c^i_j = \pm \sqrt{(\sigma^i_j)^2 - \log(\xi_D)} \quad (Eq. 6.4)$$

With the intersection points computed, the length of the overlapping region is computed
by finding the difference, $\lambda_i = Int_{i+1, L} - Int_{i, R}$, when $\lambda_i > 0$. The penalty function is
thus:

$$P_D = \frac{1}{N_{t}} \sum_{i=1}^{N_t} \frac{1}{N_{o,v}} \sum_{h=1}^{N_{o,v}} \frac{\lambda_{i,h}}{|\kappa_i|} \quad (Eq. 6.5)$$

$N_{ov}$ represents the number of overlaps, which is one less than the number of MFs. The
second penalty function, $P_C$, is similar to the first, but instead calculates the percentage of
the universe of discourse not covered by at least $\xi_C$.

### 6.4.4 Total Fitness Function

The three goals are aggregated using a weighted sum, to attempt to normalize the
components and not weight any one particular goal too heavily. The weight value is
currently set to 0.6 to effectively balance the MSE and penalty functions, and would be
adjusted by the experimenter depending on the data set used.
\[ Fitness = \frac{1}{MSE + \text{weight} \cdot (P_D + P_C)} \quad \text{(Eq. 6.6)} \]

### 6.5 Consequent Parameter Calculation

In lieu of encoding the consequent parameters, the problem can be simplified by estimating the consequent parameters via the pseudo-inverse method, common in current literature (Jang, 1993), (Russo, 2000), (Kim, Kim, & Lee, 2006). This is implemented by first calculating the firing strength of each rule (for each training example), which is defined as the product of the evaluated antecedents over the training data.

This rule strength of the \( i \text{th} \) rule is denoted by \( \tau_k \) and is calculated as follows:

\[ \tau_k = \prod_{i=1}^{N_i} \mu_{A_j}(x_i) \quad \text{(Eq. 6.7)} \]

This same firing strength is also computed in the second layer of ANFIS, and normalized in the third layer (Jang, 1993). The Weighted Mean Defuzzification is shown below.

\[ \hat{y} = \frac{\sum_{k=1}^{N_R} \tau_k y_k}{\sum_{k=1}^{N_R} \tau_k} \quad \text{(Eq. 6.8)} \]

After normalizing the firing strengths by summing them and dividing each by the sum of the firing strengths for that data example, as in Equation 6.8 above, the pseudo-inverse method is used to determine the best consequent parameters for the given training data, similar to the least-squares estimate used in ANFIS. The FIS output, \( \hat{y} \), and the desired output from the training data, \( y \), should ideally match. Thus, linear algebra can be used
to compute the optimal consequent parameters, given the antecedents. Although more computationally intensive, the pseudo-inverse is superior in reducing the training error than the standard least-squares estimate (Kim, Kim, & Lee, 2006).

6.6 Selection

The selection scheme is developed with care to ensure the better chromosomes are chosen as parents, and so that the best, or elite, chromosome remains intact. To do this, the chromosomes are evaluated for fitness, and then ranked accordingly. Then, half of the new population is created by genetic operations (crossover and mutation) from the fittest 30% of the prior population. The remaining 70% of the population is used to create the other half of the new population (Kim, Kim, & Lee, 2006). Simply selecting parents with a probability proportional to their fitness tends to skew the selection to highly favor the upper few chromosomes, and can lead to premature convergence upon a local optimum instead of the global one.
Figure 6-5 illustrates the selection scheme for the algorithm. When a pair of parents is selected as described above, they are (possibly) combined using crossover, as described in section 6.7. Once the new population (generation x+1 in Figure 6-5) has been created, elitism is employed, wherein 10% of the new population is replaced with the elite member from the previous generation (step 3). This is illustrated in Figure 6-6 Mutation and Elitism.
The fourth step is mutation, wherein the new chromosomes are independently modified by the function described in section 6.8. Finally, one chromosome is replaced again with the elite member of the prior generation, to ensure a monotonically increasing fitness level.

### 6.7 Crossover

Crossover is the genetic operation that makes two new chromosomes from two parent chromosomes (of the prior generation) by exchanging some of the genes (or parameters). There are several ways to exchange this genetic information, from simply choosing random genes from either parent, to performing an arithmetic combination of the parents’ genes. Here, three crossover schemes are investigated for suitability in training the FIS.
The first crossover scheme used here is fairly straightforward, in that a single crossover point is randomly determined (from uniform distribution) in the chromosome. This is restricted to a membership function boundary, however, and not a parameter. After the crossover point, the membership functions are swapped, though they are still mapped to their respective inputs.

Figure 6-7 Single-Point Crossover

The second crossover method is like the first, but with N crossover points. Each gene is chosen from either the first or second parent. However, all associated MFs from an input are carried over together (all crossover points are on input boundaries). This is illustrated in Figure 6-8 below.
The third and final crossover scheme considered here is similar to BLX-α (Herrera, Lozano, & Verdegay, 1998). This is an arithmetic crossover that allows a child gene to be located outside of the boundaries of its parents’ genes. Here, a random number, $\lambda$, is uniformly drawn from the interval [-0.5 1.5], as in (Kim, Kim, & Lee, 2006). The two child genes are created from the parent genes by:

\[
\text{(Eq. 6.9)}
\]

\[
\text{(Eq. 6.10)}
\]

These child values for the center parameters produced in the arithmetic crossover scheme are bounded by the values for their respective inputs, from the training data. This helps ensure adequate coverage in the actual universe of discourse (though one could imagine bad parameters would be removed due to poor fitness values).

The results of the different crossover methods are combined with the different mutation operators and detailed in the results section.
6.8 Mutation

Three mutation operators are explored in this work, combined with the three crossover operators. The first scheme models that of (Kim, Kim, & Lee, 2006), wherein the mutation variance decreases with each subsequent generation. This has the intent of narrowing in on the local optima.

\[
P_{\text{new}} = P_{\text{old}} + N(0, \rho \times \exp (-\frac{t}{50}))
\]

(Eq. 6.11)

Here, \( N(0, x) \) is a normal distribution with mean 0 and standard deviation \( x \). \( t \) is the generation number. \( \rho \) is a design parameter, here set to 0.5.

The second mutation operator in this work is derivative of that found in (Russo, 2000). Here, the amount the parameters are allowed to vary depends on the standard deviation of the MSE of the chromosomes in the population. The goal is to promote a certain level of genetic difference amongst the various FISs in the population. To do this, a value, \( \delta \), is initialized to 0.8 at the beginning of the training process. At the beginning of each generation, during the fitness evaluation, \( \delta \) is potentially modified based on the ratio of the standard deviation of the RMSE of the FISs in the population to their mean RMSE.

\[
\delta_{\text{new}} = \begin{cases} 
\delta_{\text{old}} \times 0.7 & |\sigma_{\text{RMSE}}| \geq \overline{\text{RMSE}} \times 0.5 \\
\delta_{\text{old}} \times 1.4 & |\sigma_{\text{RMSE}}| < \overline{\text{RMSE}} \times 0.3 \\
\delta_{\text{old}}, & \text{otherwise}
\end{cases}
\]

(Eq. 6.12)

Chromosomes are randomly selected to mutate (with a probability of 0.8), and if chosen, each parameter is altered by Equation 6.13.
\( P_{\text{new}} = P_{\text{old}} \cdot (1 - \delta/2 + \text{rand} \cdot \delta) \)  

(Eq. 6.13)

Here, \( \text{rand} \) indicates a uniformly distributed random number in the interval \([0, 1]\). One difference between the legacy literature and this is that the new values are bounded to either the input range (for the center) or a pre-defined range of widths.

![Before Mutation](image1)

![After Mutation](image2)

Figure 6-9 Mutation

The third and final mutation operation is similar to the first, but with no exponential decay term. The \( \rho \) parameter here is set to 0.05, smaller than before, to prevent massive mutations which would hinder the GA’s ability to converge.

### 6.9 Elitism

Elitism is the act of replacing a chromosome in a population with the best chromosome from the last generation. This ensures that when stopping at any point during the training process, the best member of that generation is the best chromosome found to date. Here, it is done twice. The first instance is immediately following the crossover operation, where 10% of the newly formed population is replaced, at random, with the elite member.
from the prior generation. Combined with a high probability for mutation, this act helps in focusing the search on higher quality chromosomes.

The second instance of elitism is immediately following mutation, where only one member of the new population is replaced with the old elite member. This ensures that the best answer is not lost due to a mutation.

6.10 Local Search

The likelihood of a chromosome being selected for optimization with local search is proportionate to its relative fitness value. Upon selection, the local maximum (fitness) is found for the chromosome using the complex method of constrained nonlinear optimization as used by (Guin, 1968) and (Wiens, Burton, Schoenau, & Bitner, 2008). This has the effect of quickly exploring the surrounding space instead of relying on the genetic operations for a task they are not well-suited. Many researchers use gradient-descent techniques for this type of search, however the gradient is not easily calculable here (due to the complex fitness function), and can only be estimated (Russo, 2000). The complex method does not use the gradient, and therefore is well-suited to this task.

The complex search method is run for a number of generations inversely proportional to the selected chromosome’s rank (by fitness), ranging from 1 to 100 generations. The search method constrains the search space for the center parameters by a fraction of the range for their associated inputs. The constraints are also bounded to ensure sensible values.
\[
\Delta Range_i = \max(input_i) - \min(input_i) \quad \text{(Eq. 6.14)}
\]

\[
LowerBound(\sigma_m) = \max(0.9* \sigma_m, \Delta Range_i * p_{\text{min}}) \quad \text{(Eq. 6.15)}
\]

\[
UpperBound(\sigma_m) = \min(1.2* \sigma_m, \Delta Range_i * p_{\text{max}}) \quad \text{(Eq. 6.16)}
\]

\[
LowerBound(c_m) = \max(\min(input_i), c_m - |0.1*c_m|, c_m - |0.1*\Delta Range_i|) \quad \text{(Eq. 6.17)}
\]

\[
UpperBound(c_m) = \min(\max(input_i), c_m + |0.1*c_m|, c_m + |0.1*\Delta Range_i|) \quad \text{(Eq. 6.18)}
\]

### 6.11 Results

The parameters for the algorithm are detailed in Table 6.1.
The different combinations of crossover and mutation operators all performed fairly consistently, as shown in Table 6.2. Here, the training and checking error (RMSE) are shown for each combination of crossover and mutation operator. Also, the minimum RMSE and the average RMSE for 10 runs is tabulated.
Table 6.2 RMSE for Mackey-Glass

<table>
<thead>
<tr>
<th></th>
<th>Mutation With Decay</th>
<th></th>
<th>Auto-Tuning</th>
<th></th>
<th>Static Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Checking</td>
<td>Training</td>
<td>Checking</td>
<td>Training</td>
</tr>
<tr>
<td>Single-Point Crossover</td>
<td>Min</td>
<td>0.00139</td>
<td>0.00127</td>
<td>0.00140</td>
<td>0.00128</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>0.00146</td>
<td>0.00133</td>
<td>0.00149</td>
<td>0.00137</td>
</tr>
<tr>
<td>N-Point Crossover</td>
<td>Min</td>
<td>0.00139</td>
<td>0.00126</td>
<td>0.00138</td>
<td>0.00125</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>0.00148</td>
<td>0.00136</td>
<td>0.00147</td>
<td>0.00134</td>
</tr>
<tr>
<td>Arithmetic Crossover</td>
<td>Min</td>
<td>0.00140</td>
<td>0.00128</td>
<td>0.00138</td>
<td>0.00126</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>0.00144</td>
<td>0.00132</td>
<td>0.00146</td>
<td>0.00134</td>
</tr>
</tbody>
</table>

In Table 6.2, the auto-tuning mutation scheme combined with N-Point crossover scheme is highlighted, because of the combined performance on the training and checking data. It is hard to say whether or not that number is statistically significant, however.

It is interesting to note the difference between the auto-tuning mutation scheme and the other two, in the fitness plots. The fitness of the best FIS continues to increase, but the average fitness of the population will oscillate as the algorithm auto-tunes the bounds of mutation. Again, this is the desired effect for maintaining a certain level of diversity in the population.
Figure 6-10 Fitness of Decaying Mutation
Table 6.3 Comparative Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Rules</th>
<th>Train RMSE</th>
<th>Test RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEFREX (Russo, 2000)</td>
<td>20</td>
<td>0.0054</td>
<td>0.0061</td>
</tr>
<tr>
<td>ANFIS (Jang, 1993)</td>
<td>16</td>
<td>0.0016</td>
<td>0.0015</td>
</tr>
<tr>
<td>Passive MFVS (Kim, Kim, &amp; Lee, 2006)</td>
<td>24</td>
<td>No Data</td>
<td>0.0011</td>
</tr>
<tr>
<td>Proposed, without MRB</td>
<td>16</td>
<td>0.001393</td>
<td>0.001276</td>
</tr>
</tbody>
</table>

Figure 6-11 Fitness of Auto-Tuning Mutation
A run using 3 MFs per input immediately resulted in overfit, as shown in Figure 6-12.

### 6.12 Conclusions

Though improved over ANFIS, the algorithm cannot train better than the results obtained by Kim, Kim, Lee (Kim, Kim, & Lee, 2006). The primary reason for this appears to be a smaller rule base. Expanding to accommodate 3 MFs per input would result in 81 rules for this architecture, and would result in overtraining, as shown in Figure 6-10.
Using the auto-tuning mutation scheme with the N-point crossover appears to minimize the errors in a best-case scenario, though the average of 10 runs shows that it is not guaranteed to outperform the other methods every time. This supports the theory that the auto-tuning mutation scheme adequately maintains enough genetic diversity within the population to explore the universe of discourse, while fully utilizing the local search to converge upon the optimal solution. However, it also shows that there may not be a statistically significant improvement over the other crossover and mutation combinations, and that they all perform fairly well.

Due to the limitations of the framework, a mutable rule base is considered, in Chapter 7. This improvement to the algorithm attempts to optimize the number of rules in the rule base, while limiting the number of membership functions for each input.
Chapter 7

7 Mutable Rule Base

7.1 Introduction

As shown in the results of Chapter 6, the constraints of the standard algorithm do not allow for finding the optimal FIS for time series prediction. The work shown in (Kim, Kim, & Lee, 2006) used 24 rules and had a slightly smaller error than the work in Chapter 6 which used 16 rules. To expand the rule base, some constraints must be lifted.

Remember that the rules are a mapping of one membership function for each antecedent to a particular output membership function. The output membership functions are computed automatically in the calculate consequents step, so the rule information stored is simply an array of membership function indices, one for each input. A zero indicates a “don’t care” condition, and the input is ignored. For example, a rule encoded as \([2 \ 2 \ 0 \ 1]\) represents the second MF of inputs 1 and 2, along with the first MF of input 4. Input 3 is ignored in this rule. This is the encoding scheme for each rule as a gene. A rule base chromosome would be the aggregation of all of the rule genes for a fuzzy inference system rule base.
Here, the algorithm is improved by allowing for the rule base to be subjected to evolutionary optimization as well as the parameters of the membership functions. Instead of having a population of rule base chromosomes, the rule base chromosome is subject to optimization along with the membership functions chromosome. The rule base can grow in two ways. First, during the crossover phase rule bases are merged, often resulting in a child FIS with a larger rule base than that of either parent. Second, new rules can be randomly added to the rule base during the mutation step. This is discussed in the subsequent sections in more detail.
Once the rule base grows to such a large size that it overfits the training data, the testing (checking) error is likely to increase. This indicates that the FIS is no longer modeling the underlying relationship of the training data, but rather the noise present in it. Unseen examples are likely to be predicted poorly, if at all. To prevent overfitting the data, the optimum number of rules must be determined, and after reaching that number, additional weak rules should be removed. The approach to mitigate this is discussed at the end of the chapter.

Finally, the algorithmic improvements are tried out on the Mackey-Glass time series, as well as the well-studied Box-Jenkins Gas Furnace data set. The former data set illustrates the improvements on the prior methodology, while the latter (Box-Jenkins) suggests a generalization ability of the algorithm, due to its relatively small size and presence of real-world noise.
7.2 Initial Population

The initial population is created in the same manner as in Chapter 6.

An optional final step of the population initialization is to randomly prune some of the rules. Initially, a rule is created for each permutation of the membership functions, given one MF per input. The intention of removing rules is to increase the generalization ability of the FIS, along with keeping the FIS compact. During the course of training, the number of rules can and will change by increasing in the presence of underfit and decreasing in the presence of overfit. The results of this smaller initial rule base show a
slower convergence on an optimal solution, due to the need to build up the rule base to an adequate size. However, this can prove beneficial for noisier or smaller data sets that require fewer rules to adequately model the system. Also, pruning the initial rule base is useful when there are many MFs, which would result in a very large number of rules under normal conditions.

7.3 Detecting Overfit

Many machine learning techniques employ methods to reduce overfit inherently, such as cross-validation. This method could be applied here, but it reduces the number of examples in each fold of the input training data set. Also, cross-validation requires several more instances of the algorithm, which is a large computational expense.

In this work, overfit is detected, and then mitigated. Here, two counters are used to increment whenever the training error decreases, or checking error increases. A decrease in training error is good, but not at the expense of the checking error – the measure of generalization. If the training error increases, or the checking error decreases, the respective counters are decremented by two, which helps the system learn that the overfit conditions are not in effect. Once the decreasing training error counter outweighs the increasing checking error counter, overfit is said to have occurred, and mitigation begins (a combination of an additional penalty factor specified in Section 7.4 and reductions in the mutation step in Section 7.7).
Another overfit detection method employed is the case where the checking error has 20 times the training error in one generation. This is set based experimental findings, and depending on the data, may need to be adjusted.

7.4 Fitness Evaluation

The modification for fitness evaluation adds an additional penalty factor to the previous computation in Section 6.4, which is the mitigating factor for overfit conditions. The penalty factor is proportional to the difference from the number of rules in the FIS to the optimal number of rules, and is shown in Eq. 7.1.

\[
P_R = \frac{|NumberRules - optimumRules|}{maxRules - minRules} \quad (Eq. 7.1)
\]

The penalty term is weighted, so as to not completely overrule the other penalty terms, with a ratio of the checking error, squared, to the training error, for the fittest member of the population, before the local optimization occurs.

\[
Weight_R = \frac{CheckMSE^2}{TrainMSE} \quad (Eq. 7.2)
\]

A simple ratio of checking error / training error = 1 indicates no overfit, where a high checking error to training error ratio would indicate bad overfit. Squaring the checking error effectively scales the penalty term to approximately the same range as the training error term of the particular FIS being evaluated. Note that the checking and training terms in that weight are those of the fittest FIS in the population prior to local search.
With the new penalty factor in place, the fitness function is shown below.

\[
Fitness = \frac{1}{MSE + \text{Weight}_C \times (P_C + P_D) + \text{Weight}_R \times P_R}
\]  
(Eq. 7.3)

### 7.5 Selection

No changes necessary. See Chapter 6.

### 7.6 Crossover

When two parents are combined with a crossover operation, it is also necessary to combine their rule bases. Very simply, the two children receive rules from the parent rule bases at random, with each rule going to one of the two children. This is the same as an n-point crossover scheme for the chromosome-encoded rule bases. Following this operation, the rules are checked for validity, since differing numbers of membership functions can lead to invalid rule mappings (for antecedents that do not exist). Invalid rules are discarded.

Finally, the membership functions for each input are sorted based on the center parameter, so the rule mappings semantically mean the same thing. That is, the membership function indices must be the same from FIS to FIS in order for the rule...
mappings to maintain consistency. Reordering the MFs based on the center parameter help preserve that mapping. An index of 1 always maps to the lowest center value.

7.7 Mutation

The membership functions are mutated using the auto-tuning mutation operator in Section 6.8. Additionally, the rule base undergoes a mutation-like operation wherein rules are randomly modified via remapping, removed altogether, or added.

After the membership function mutation, the rules are iterated over, and for each input, with the probability of 0.4, the mapping will either change from one of the membership functions to “don’t care”, or from “don’t care” to a randomly chosen membership function for that input. This is similar to a binary mutation scheme wherein bits (genes) are flipped between 0 and 1.

If overfit has been detected, a rule is randomly removed with a probability of 0.4. This helps reduce the size of the rule base. On the flipside, if overfit has not been detected, there is a 50% chance that five potential rules are generated with random mappings. These rules are added to the rule base if they are not duplicates of already existing rules. This helps grow the rule base to find the optimum number of rules to properly model the data.

7.8 Elitism
7.9 Local Search

No changes necessary.

7.10 Rule Base Pruning

In the event that overfit does occur, it is helpful to help remove rules that may not be as useful as others, due to 1) low firing strength, 2) not covering many inputs, or 3) having a high mapping similarity to a stronger rule. After each genetic operation the number of rules in the newly modified FIS is checked against the maximum number of rules allowed. If the number of rules is more than the current maximum allowable rules, the rules are ranked by their firing strength. While there are too many rules in the rule base, the weakest rule is either removed or merged. It is removed if it has a firing strength of less than 0.001. If it has a firing strength higher than that, as it often does, then it is merged with the most similar rule.

The difference between two rules is measured by taking the antecedents that are not mapped to 0 in either case, and summing the differences between the membership function indices. For example, rules of [1 0 1 2] and [1 1 1 1] would have a difference of 1. Ignoring “don’t care” antecedents helps to merge these rules with more useful ones. In this case, the rules would be merged by replacing the 0 with the antecedent mapping from the other rule. For non-zero antecedents, the stronger rule takes precedence. If [1 0
1 2] is the stronger rule, the result would be [1 1 1 2], whereas if [1 1 1 1] is the stronger rule the result would simply be [1 1 1 1].

The rule bases are slowly optimized inherently by the genetic algorithm due to the new penalty factor; however the rule pruning helps the GA converge quicker.

### 7.11 Results on the Mackey-Glass Time Series

The results from the mutable rule base additions indicate lower training and checking error than before, which indicate a good solution that should have good performance on future, unseen, examples.

The algorithm enhancements are tested using roughly the same parameters as in Chapter 6, except 100 generations (which are necessary due to the increase in search space dimensionality), and only using the N-Point Crossover and Auto-Tuning mutation schemes, since they performed slightly better than the other crossover / mutation combinations. An FIS is obtained with a training RMSE of $6.96 \times 10^{-4}$ and checking RMSE of $6.94 \times 10^{-4}$, with 27 rules in the rule base. This is a larger rule base than the one in (Kim, Kim, & Lee, 2006) by 3 rules, but it is the lowest error of the algorithms in the research for a trained FIS.
Table 7.1 Results for Modifiable Rule Base

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Rules</th>
<th>Training RMSE</th>
<th>Test RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEFREX (Russo, 2000)</td>
<td>20</td>
<td>0.0054</td>
<td>0.0061</td>
</tr>
<tr>
<td>ANFIS (Jang, 1993)</td>
<td>16</td>
<td>0.0016</td>
<td>0.0015</td>
</tr>
<tr>
<td>Passive MFVS (Kim, Kim, &amp; Lee, 2006)</td>
<td>24</td>
<td>No Data</td>
<td>0.0011</td>
</tr>
<tr>
<td>Proposed, without MRB</td>
<td>16</td>
<td>0.001393</td>
<td>0.001276</td>
</tr>
<tr>
<td>Proposed, with MRB</td>
<td>27</td>
<td>0.000696</td>
<td>0.000694</td>
</tr>
</tbody>
</table>
Surprisingly, most of the rules contained one or more “don’t care” mappings. The rules, shown below in Table 7.2, contain 4 of the original 16 rule mappings. This is, in part, due to the implementation of the mutation operation on the rules, often setting the values to “don’t care”.

Table 7.2 FIS Rule Mappings

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
<th>Input 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Another well-known time series is the Box-Jenkins Gas Furnace data set, discussed in Section 5.2. Due to the presence of noise and fewer examples in this set, it is easy to overfit the data. One way to handle this in ANFIS is to purposefully overtrain the FIS, noting where the checking error begins to rise. Subsequently re-train the FIS for that number of generations (epochs). Several tests are run on ANFIS to determine its ability
to model the Box-Jenkins data. From the 292 examples, the training data are the first
148, and the remaining 144 constitute the checking data.

<table>
<thead>
<tr>
<th>#MFs/Input</th>
<th>Type</th>
<th>#Generations</th>
<th>Training Error</th>
<th>Checking Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Mamdani</td>
<td>200</td>
<td>0.11581</td>
<td>0.56416</td>
</tr>
<tr>
<td>2</td>
<td>Sugeno</td>
<td>2</td>
<td>0.08483</td>
<td>0.56060</td>
</tr>
<tr>
<td>3</td>
<td>Mamdani</td>
<td>300</td>
<td>0.06821</td>
<td>0.65492</td>
</tr>
<tr>
<td>3</td>
<td>Sugeno</td>
<td>2</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

In Table 7.3 above, the Sugeno-style FIS with 3 MFs per input had more trainable
parameters than training data examples, and the algorithm could not properly train the
FIS.

Using the same data sets, the Mutable Rule Base Hybrid Genetic Algorithm is used to
train the FIS. The GA parameters are listed below, which are the same as in Section
6.11.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Number of Generations</td>
<td>50</td>
</tr>
<tr>
<td>Population Size</td>
<td>100</td>
</tr>
<tr>
<td>Number of MFs per Input</td>
<td>2</td>
</tr>
<tr>
<td>Probability of Crossover</td>
<td>0.7</td>
</tr>
<tr>
<td>Probability of Mutation</td>
<td>0.9</td>
</tr>
<tr>
<td>Overlap Penalty Weight</td>
<td>0.6</td>
</tr>
<tr>
<td>Overlap Threshold</td>
<td>0.6</td>
</tr>
<tr>
<td>Discontinuity Threshold</td>
<td>0.3</td>
</tr>
</tbody>
</table>
In Figure 7-4 above, the checking error increases drastically during the initial training stages, but then tapers off as the overfit is mitigated via rule reduction. The best result, shown with the circle at generation 22 had a training RMSE of 0.124 and a checking RMSE of 0.500, and used five rules. That is a marked improvement over the ANFIS results on the same data set. The five rules are shown in
Table 7.5.
Table 7.5 Box-Jenkins Rule Mappings

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
<th>Input 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

7.13 Conclusions

Using the mutable rule base helps model the data more accurately by optimizing the structure of the FIS alongside the membership function parameters. The sacrifice made is a larger rule base at the expense of a low error rate (for the Mackey-Glass Time Series). However, this is a sacrifice many people would want to make.

The results seem to indicate that there is either a correlation between number of rules and training error, or an added utility in rules with “don’t care” antecedents. Obviously an increase in rules leads to additional training parameters which can better model the training data. One could also reason that “don’t care” antecedents indicate a less-useful input value. However, perhaps it is actually showing that there are rules that model relationships of inputs “working together” as well as independently. Either way, much
lower training and checking errors are obtainable than reported before, which supports the usefulness of the increased rule base size for the Mackey-Glass Time Series.

For the Box-Jenkins Gas Furnace Data, the rule base shrunk from a default of 16 rules down to merely 5. The overfit detection worked as intended and provided a more generalized solution than ANFIS. Different papers report training on this data set, but with different inputs and different partitioning of the training versus checking data sets. Due to this, it is difficult to compare results of many of the other works.
Chapter 8

8 Recommendations for Future Work

8.1 Improved Overfit Detection

The method for detecting and mitigation overfit in this paper is not the focus of the work, and could be improved. It is challenging to determine what a significant increase in checking error could be, and whether or not it is indicative of overfit. One way to circumvent this detection is actually prevention, which can be done using N-Fold validation or only training on a random subset of the training data during each generation. This poses certain challenges itself, as studies would need to be done in how to best implement that in both the global and local optimization heuristics.

8.2 Consequent Parameter Calculation

Some quick experimentation shows that the consequent parameter calculation can model the noise, resulting in overfit, and that using more robust linear regression methods can mitigate the tendency to find an exact match to the input data. The robust multilinear regression schemes in use here iteratively weight the outliers to lessen their effect on the resultant consequent parameters. Some of the methods include Andrews, bisquare, fair,
and Welsch. The initial results of this endeavor were too generalized to produce a FIS with low training or checking error, with the best one at 0.40 and 0.87 for training and checking RMSE, respectively.

8.3 Additional Antecedent Membership Functions

Another idea for future exploration is architecting the FIS to have several membership functions available for each input, each with a smaller grid partition. The rule base could be approximately the same size as presented in Chapter 7, but with fewer rules mapping to “don’t care” conditions. Since several more options are possible in this setup, the curse of dimensionality may have a prohibitive effect on the number of computations. Another potential downside to this approach is the smaller membership functions would either overlap each other more or have a lower firing strength due to lack of input space coverage.

8.4 Support for Mamdani-Style FIS

Another way to potentially reduce the overfit or over-training of the FIS on smaller training data sets is to use a Mamdani FIS with only one consequent parameter per rule, instead of a linear function of several consequent parameters. This consequent parameter could still be computed, or trained genetically. However, due to the reduction in number of system variables, the resultant FIS could have an increase in generalization. This approach would require significant development to the existing code base, but
theoretically can have a great impact on generalization without the increase in computational complexity.

8.5 Use of Fuzzy Clustering

Currently, grid partitioning is used to provide clear boundaries for the center values of the input membership functions. However, fuzzy clustering can be used to create the initial FIS population without these rigid boundaries. In fact, the MFs may better represent the input data if such an approach were used. In order for there to be any diversity within the population, however, the number of clusters to find (the number of membership functions assigned to each input) should be varied often, for each FIS in the population. The additional downside to this approach is the lack of order and constraints associated with the grid portioning system. It would be interesting to try this method with the suggested algorithm enhancement listed in Section 8.3, with several potential membership functions.
References


