The Third AFSS(1998), 613-618

Fuzzy Logic Controller Design via Messy Genetic Algorithm

Oh-Kook Kwon*, Wook Chang*, Young-Hoon Joo**, Jin-Bae Park*

* Dept. of Electrical Engineering, Yonsei University, Seoul, Korea

Tel:+82-2-361-2773; Fax: +82-2-392-4230; E-mail: jbpark@bubble.yonsei.ac.kr

** Dept. of Control and Instrumental Engineering, Kunsan National University, Kunsan, Korea

Tel:+82-654-469-4706; Fax: +82-654-466-2086; E-mail: yhjoo@ks.kunsan.ac.kr

Abstract

The success of a fuzzy logic control system solving any given problem critically depends on the architecture of the network. Various attempts have been made in optimizing its structure its structure using genetic algorithm automated designs. In a regular genetic algorithm, a difficulty exists which lies in the encoding of the problem by highly fit gene combinations of a fixed-length. This paper presents a new approach to structurally optimized designs of a fuzzy model. We use a messy genetic algorithm, whose main characteristic is the variable length of chromosomes. A messy genetic algorithm is used to obtain structurally optimized fuzzy models. Structural optimization is regarded important before neural network based learning is switched into. We have applied the method to the example of a cart-pole balancing.

Keywords : fuzzy control, messy genetic algorithm, automated design

1. Introduction

There is a great number of current interest in developing the design of fuzzy logic controllers using genetic algorithms[1][2]. The fuzzy logic controllers are very effective for controlling complex and poorly defined systems as it incorporates the knowledge of human experts to achieve good control strategies. Once the controller structure is determined, the key elements influencing the performance of the fuzzy logic control are the rules, scaling factors and shapes of the membership functions[3]. The knowledge of experts has influence upon the design of the controllers, especially the structure of controllers.

In this paper we introduce the design method of the fuzzy logic controllers using messy genetic algorithms. Our method differs from other approaches in two special phases. Firstly, we use a messy genetic algorithm[4] which process the variable length strings, in contrast to standard genetic algorithms which work with a fixed length coding scheme. Messy Genetic Algorithms therefore allow a flexible representation of fuzzy rules in the controllers rulebase[5]. Secondly, a backpropagation learning algorithm is used to tune finely the parameters of fuzzy membership functions. In short, after obtaining the fuzzy rules of the controller by messy genetic algorithms then we tune finely the fuzzy membership functions by backpropagation learning algorithms. We demonstrate an application of our method in teaching a controller for a cart-pole balancing problem.

2. Messy Genetic Algorithm

Genetic algorithms are optimization methods which are used to process a population of strings by applying genetic operators such as selection, recombination and mutation. Central to evolutionary system is the idea of a population of genotypes that are elements of high dimensional search space. More generally, a genotype can be thought of as an arrangement of genes, where each gene takes on values from a suitably defined domain of values[6]. Solutions of the optimization space are coded as fixed-length, fixed-locus strings defined over an alphabet of alternatives at each position. Genetic algorithms are particularly well suited for nonlinear fitness functions with many local maxim, where the overall fitness can not be decomposed into contributions from single genes[5].

Genetic Algorithms have been successfully applied to many fuzzy control applications, but not without objections. The problem arises with the encoding of the problem parameters. In a regular genetic algorithm, a coded chromosome is in fixed-length that highly fit allele combinations are formed to obtain a convergence towards global optima. Unfortunately the required linkage format (or the structure of the controller to be coded) is not exactly known and the chance of obtaining such a linkage in a random generation of coded string is poor. Poor linkage also means that the probability of disruption on the building block by the genetic operators is much higher[7]. Although inversion and reordering methods can be used to adaptively search tight gene ordering, .these are too slow to be considered useful.

The new learning method proposed uses messy genetic algorithm[4,5,6]. The main differences between a messy genetic algorithm and a regular genetic algorithm are as follows;

- A messy genetic algorithm uses varying string lengths
- The coding scheme considers both the allele positions and values
- The crossover operator is replaced by two new operators called "cut and splice"
- It works in two phases - primordial phase and juxtapositional phase.

2.1. Coding and Decoding

In a messy genetic algorithm, the code of integer is more efficient than that of binary[3]. Here, one parameter uses one coding variable and hence dramatically reduces the memory usage. This ensures that the string length is kept to a minimal and speeds up evolutionary operations, while also reducing the unnecessary inner parameter disruptions caused by crossover and mutation.

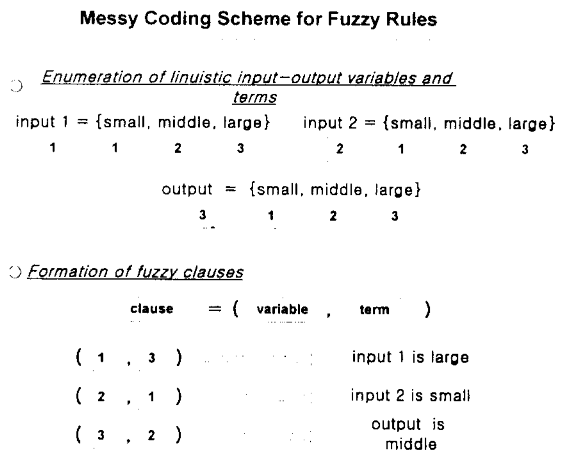
Messy Coding Scheme for Fuzzy Rules
Enumeration of linuistic input-output variables and terms
input 1 = {small, middle, large}
1 1 2 3
input 2 = {small, middle, large}
2 1 2 3
output = {small, middle, large}
3 1 2 3
Formation of fuzzy clauses
clause = (variable , term)
(1 , 3)
input 1 is large
(2 , 1)
input 2 is small
(3 , 2)
output is middle

Figure 1. code of a fuzzy clause for a messy GA

A messy genetic algorithm is accomplished first by defining a messy gene as an ordered pair that identifies the gene by its name(or index) and value and then by defining a messy chromosome as a collection of messy genes. For example, the set (2,3) would correspond to the second gene with value '3'. Another feature of a messy GA is that the order of the string is irrelevant, i.e. the string {(1,3)(2,1)(3,2)} and the string {(2,1)(3,2)(1,3)} are identical. Notice that we have not required all genes to be present, nor have we precluded the possibility of multiple, possibly contradictory, genes. For example, for a three parameter problem the strings {(1,1)(2,1)} and {(1,1)(2,1)(3,3)(2,2)} are both valid. In the first case the string is said to be under-specified because there is no gene 3, and in the second case the string is said to be over-specified because gene 2 appears twice.

2.2. Messy Operators

To handle strings of variable length, the standard crossover operator is no longer suitable. Instead it is replaced by two new operators, which is called *cut and splice*. A schematic diagram of the cut and splice operator is displayed figure 2. The cut operator simply cuts the string in two parts at randomly chosen position. The splice operator concatenates two strings, which could have been previously cut, in a randomly chosen order. When the cut and splice operators are applied simultaneously to two parent strings they alt in a similar way to the ordinary crossover operator.

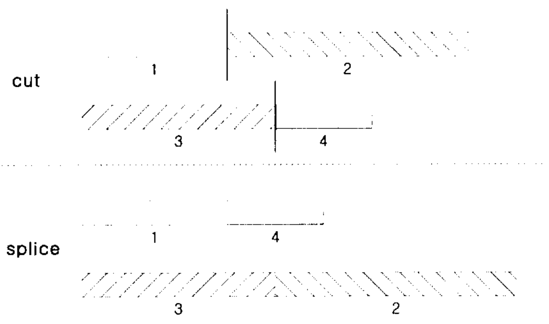

Figure 2. A messy cut and splice operation

In messy genetic algorithms the positions of cuts in strings, which are to be joined can be chosen independently, whereas in regular genetic algorithms the crossover points must coincide.

The selection mechanism is as in regular genetic algorithm but executed in primordial and juxtapositional phases. During the primordial phase, the population is first initialized to contain the all possible building block of a particular length, thereafter only the selection operator is applied. This results in enriched population of building blocks whose combination will create optimal or near optimal strings. And the population size is reduced by halving the number of individuals at specified intervals. The juxtapositional phase follows the primordial phase, and here the genetic algorithm invokes the cut and splice and the other operators.

3. Fuzzy Logic Control

3.1. Fuzzy Inference System

A *i*th fuzzy if-then rule assumes the form

$$R^i : \text{If } x_1 \text{ is } A_{1i} \text{ and } x_2 \text{ is } A_{2i}, \text{ then y is } B_i$$

where x_1 , x_2 and y are input and output variables, respectively. A_{1i}, A_{2i}, and B_i are linguistic terms characterized by fuzzy membership function $\mu_{A_{ji}}(x_{ji})$ and $\mu_{B_i}(y_i)$, respectively.

We choose a Gaussian-type membership function with a bell shaped form as follows:

$$\mu_{A_{ji}} = \exp\left(-\frac{(x_{ji} - c_{ji})^2}{a_{ji}^{\;2}} \right) \qquad (1)$$

where $\{a_{ji}, c_{ji}\}$ are the parameters of the membership function. In a simplified fuzzy inference, the weight of the consequent part, $\mu_{B_i}(y_i)$, is a constant w_i.

We describe formulas between input and output values of the fuzzy inference. The output value y_k, with respect to the kth input pattern is expressed by

$$\mu_i = \prod_{j=1}^{M} \mu_{A_{ji}} \tag{2}$$

$$y_k = \frac{\sum_{i=1}^{N} \mu_i \cdot w_i}{\sum_{i=1}^{N} \mu_i} \tag{3}$$

where M is the number of inputs, which is the dimension of input space, N the number of fuzzy rules, μ_i the fitness in the ith fuzzy rule, $\mu_{A_{ji}}$ the membership function, and w_i the interconnection weight.

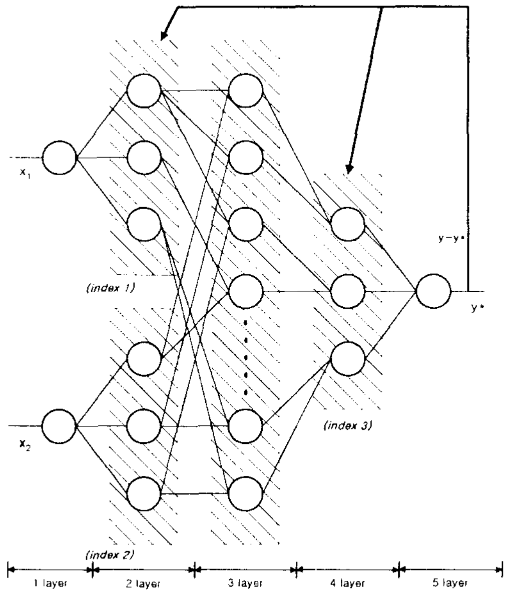

Figure 3. The structure of fuzzy inference

3.2. Learning Algorithm

The learning algorithm is based on an adaptation of the backpropagation learning method which mimic fuzzy inferencing and defuzzification. The parameters to learn are the weights, which relate to the shape of the membership functions.

Figure 4 shows the coding of fuzzy rule base for a sample gene.

The fine tuning procedure updates the parameters acquired by the genetic algorithm using a back-propagation algorithm. The general form of back-propagation is

$$w[n+1] = w[n] + \eta \cdot \left(\frac{\partial E}{\partial w} \right) \tag{4}$$

where w is the weight, E the error function, and the learning rate.

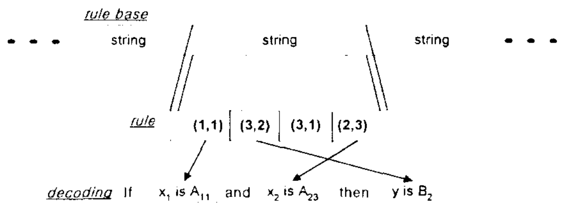

Figure 4. Coding of fuzzy rules in string

4. Simulation

We apply our method to the inverted pendulum system as a simple example of a nonlinear control system. Figure 5 shows an inverted pendulum system(or called a cart-pole system), which is a classic example of a nonlinear feedback control system. A rigid pole is attached to a cart with a hinge, a free joint with only one degree of freedom. The cart can move to the right or left on rails when a force is exerted on it. This dynamic system is characterized by two state variable.

The control goal is to find the applied force u as a function of the state variable x:

$$x = [\theta, \dot{\theta}] \qquad (5)$$

where θ is the pole angle with respect to the vertical axis and $\dot{\theta}$ is angular velocity of the pole. These state variables are governed by the following equation[8].

$$\dot{\theta} = \frac{g\sin\theta - \cos\theta\left[F + m_g l(\sin\theta)^2\dot{\theta}^2\right]}{\frac{4}{3} - \frac{m_g l(\cos\theta)^2}{m_c + m_g}} \qquad (6)$$

where $g = 9.8[m/\sec^2]$ for the constant of gravity, $m_g = 1.0[kg]$ for the weight of the pole, $m_c = 1.0[kg]$ for the weight of the cart, $l = 0.5[m]$ for the length of the pole and F is the pulling force.

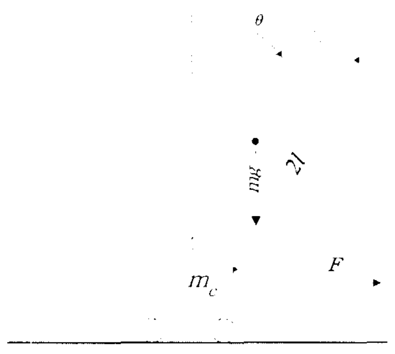

Figure 5. The inverted pendulum system

Figure 6 shows the resulting structurally optimized fuzzy inference system. Figure 7 shows the response of the pendulum for various initial conditions outside the ones used for optimization for the best final solution. We can see that with the messy genetic algorithm optimization the pendulum settles to the vertical position very quickly.

5. Conclusion

In this paper has presented a composite method of fuzzy logic control and messy genetic algorithms. The main characteristic of messy genetic algorithms is the variable length of chromosomes. A messy genetic algorithm is used to obtain structurally optimized fuzzy models. Structural optimization is regarded important before neural network based learning is switched into. After the structural optimization is determined using messy genetic algorithms, the parameters of the fuzzy membership functions are adjusted using the backpropagation learning algorithm. It has been successfully applied to a cart-pole balancing problem.

angle

angle velocity	NB	NS	NZ	PZ	PS	PB
NB			NB	NS		
NS			NS	NZ	PS	
NZ		NS	NZ	PZ	PZ	PS
PZ	NB	NZ	NZ	PZ		PB
PS	NS			PS		
PB			PS		PB	

Figure 6. The optimized fuzzy rule base

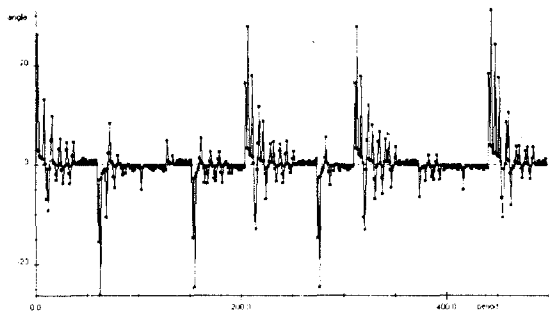

Figure 7. The trajectory of the output

References

[1] C. L. Karr, "Design of a Cart Pole Balancing Fuzzy Logic Controller Using a Genetic Algorithm", SPIE Conference on Applicaitons of Artificial Intelligence, Bellingham, WA, 1991.

[2] J. L. Castro, M. Delgado and F. Herrera, "A Learning Method of Fuzzy Reasoning by Genetic Algorithms", EUFIT 93, Vol. 2, pp. 804-809, Aachen 1993.

[3] Y. Li and K. C. Ng, "A Uniform Approach to Model Based Fuzzy Control System Design and Structural optimisation", Genetic Algorithms and Soft Computing, F. Herrera and J. L. Verdegay(Eds.), Physica-Verlag Series 'Studies in Fuzziness', 8, pp. 129-151, 1996.

[4] Kalyanmoy Deb, David E. Goldberg, "mGA in C: A Messy Genetic Algorithm in C", IlliGAL Report No. 91008, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, September 1991.

[5] Frank Hoffmann, Gerd Pfister, "A New Learning Method for the Design of Hierarchical Fuzzy Controllers Using Messy Genetic Algorithms", IFSA 95, July 1995.

[6] M. Chowdhury and Yun Li, "Messy Genetic Algorithm Based New Learning Method for Structurally Optimized Neurofuzzy Controllers", IEEE International Conference on Industrial Technology, Shanghai China, December 1996.

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

[8] J.S.R. Jang, C.T. Sun and E. Mizutani, *Neuro-fuzzy and Soft Computing*, Prentice-Hall International, NJ, 1997.

[9] Y.H. Joo, H.S. Hwang, K.B. Kim and K.B. Woo, "Fuzzy System Modeling by Fuzzy Partition and GA Hybrid schemes", Fuzzy Sets and Systems 86, pp. 279-288, April 1997.

[10] Y.H. Joo, H.S. Hwang, K.B. Kim and K.B. Woo, "Linguistic model identification for fuzzy system", IEE on Electronic Letters Vol. 31, pp. 330-331, February 1995.