Messy Genetic Algorithm Based New Learning Method for Structurally Optimised Neurofuzzy Controllers

M. Munir-ul M. Chowdhury
Centre for Systems and Control and
Dept. of Electronics and Electrical Engineering
University of Glasgow, Rankine Building,
Glasgow G12 8LT, U.K.
E-mail: chy@elec.gla.ac.uk

Yun Li
Centre for Systems and Control and
Dept. of Electronics and Electrical Engineering
University of Glasgow, Rankine Building,
Glasgow G12 8LT, U.K.
E-mail: Y.Li@elec.gla.ac.uk

Abstract - The success of a neurofuzzy control system solving any given problem critically depends on the architecture of the network. Various attempts have been made in optimising its structure using genetic algorithm automated designs. In a regular genetic algorithm, however, a difficulty exists which lies in the encoding of the problem by highly fit gene combinations of a fixed-length. For the structure of the controller to be coded, the required linkage format is not exactly known and the chance of obtaining such a linkage in a random generation of coded chromosomes is slim. This paper presents a new approach to structurally optimised designs of neurofuzzy controllers. Here, we use messy genetic algorithms, whose main characteristic is the variable length of chromosomes, to obtain structurally optimised FLC. Structural optimisation is regarded important before neural network based local learning is switched into. The example of a cart-pole balancing problem demonstrates that such an optimal design realises the potential of nonlinear proportional plus derivative type FLC in dealing with steady-state errors without the need of memberships or rule dimensions of an integral term.

Keywords: Fuzzy Control, Neurofuzzy Control, Messy Genetic Algorithms.

I. INTRODUCTION

By far the most successful application of fuzzy logic (FL) [1] has been to systems and control, and this is termed fuzzy logic control (FLC) [2]. It is 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 FLC are the rules, scaling factors and shapes of the membership functions (MFs) [3]. By carefully choosing the parameters of the fuzzy controllers, it is always possible to design a FLC that is suitable for the non-linear system under consideration.

However, the main drawback in conventional designs is their dependence on the human experience and, in particular, that on the choice of the controller structure. Other limitations of conventional FLC designs are that they can be tedious, trial and error and unadaptive. A number of hybrid techniques, such as artificial neural networks (ANNs) and genetic algorithms (GAs), have been employed to tackle some of these problems in the past decade [3,4].

In this paper, a new approach to globally optimal design of the NFCs is proposed based on messy genetic algorithms (mGAs). Structural Optimisation is achieved by the flexible encoding mechanism of the mGA, before on-line adaptation using ANN learning. The layout of the remainder of the paper is as follows. Section 2 outlines the type of neurofuzzy structure employed, while section 3 gives an overview of mGA. The optimisation and learning algorithm is described in Section 4. The techniques are illustrated with an example in Section 5 and we conclude in Section 6.

II. NEUROFUZZY STRUCTURE

An ANN is a network of highly interconnected elements, or neurons, which is structurally similar to the biological nervous system in the brain, and thus enabling the whole network to function in a similar manner to the brain. The main feature of ANNs is their ability to learn from examples. This is achieved by adjusting the strengths, or weights, of the interconnections according to some learning rule that can be supervised or unsupervised. An important feature of an ANN is that it is a universal model and also an integrated controller. The network adapts to and represents the real world by direct input/output (I/O) mapping. This is in common with fuzzy logic [4]. It is thus natural and feasible to map an FLC onto an ANN to form a neurofuzzy controller (NFC) and exploit the adaptive capabilities while inferencing by rules.

Of all the schemes used to integrate the learning abilities of ANNs with FLC systems, the most widely

ISBN 0-7803-3104-4

used is one in which the fuzzy system is installed in an ANN architecture akin to a multilayered neural network. Each node of the network performs a function such as to make the entire network equivalent to the fuzzy system. In this approach, the gradient descent method that is similar to the backpropagation algorithm is used. Fig. 1 shows the structure of the neurofuzzy controller. This NFC is essentially a connectionist model in the form of a multilayered feedforward network.

In this architecture x_i is the input state vector. Layers (1) to (3) represent the premise part and layers (4) and (5) represent the consequent part of the rules. The Mamdani min-max inferencing is used and so neurons with ($\wedge$) indicate a **min** operation and ($\vee$) indicate a **normalising** operation. The node $a(i)$ and $b(i)$ are the fuzzy sets of the inputs and $d(i)$ the fuzzy sets of the output. The defuzzification method used to obtain the crisp output y is the centre of gravity (COG). Since the network essentially represents a FLC mapping there are restrictions on how much the network can be adjusted in order to achieve the desired actions from the systems, e.g. the number of layers cannot be altered since this has direct relation to the inferencing mechanism. This limits the structural optimisation to the type of activation function of the neurons; the number of neurons per layer and the necessary links between adjacent layers.

III. MESSY GENETIC ALGORITHMS

Genetic algorithms are loosely modelled on processes that appear to be at work in biological evolution and the working of the immune systems. Central to evolutionary system is the idea of a population of

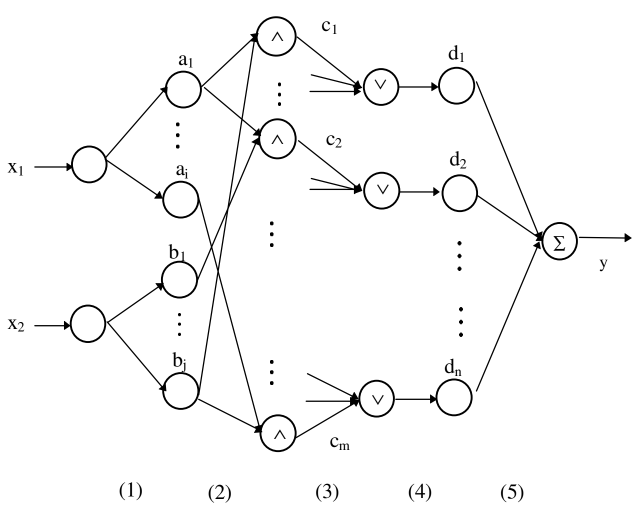

Fig. 1 Structure of an NFC

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. Each genotype encodes for typically one, but possibly a set of candidate solutions, *phenotypes*, - in our case a class of neuro-fuzzy architecture. The evolutionary process works on a population of such genotypes, preferentially selecting genotypes that code for high fitness phenotypes and reproducing them. Genetic operators such as *mutation*, *crossover*, *inversion*, etc., are used to introduce variety into the population and to sample variants of candidate solutions represented with in the current population. Thus by survival of the fittest GA over several generations, the population gradually evolves towards genotypes that correspond to high fitness phenotypes. A GA is a nondeterministic search algorithm based on the ideas of genetics. GAs try to mimic the Darwinian theory of natural selection and evolution, tending to find optimal solutions to problems instead of trying to solve them directly.

GAs are global optimisation methods requiring no derivative information and 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 GA, 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 [1]. 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 GA [5,6]. The main difference between an mGA and a regular GA is that the mGA 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;* and it works in two phases - *primordial* phase and *juxtapositional* phase.

A. Coding and Decoding

In this paper, efficient integer [3] (as opposed to binary) coding is used in the mGA. 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 [3].

In the original mGA, each gene is a set of numbers which indicates the gene's index, and its value. For example, the set **(2,4)** would correspond to the second gene with value '4'. Another feature of mGA is that the order of the string is irrelevant, i.e. the strings **((2,3) (3,1) (1,3))** and **((3,1) (2,3) (1,3))** are identical. It is also possible for a string to not have the full gene complement. 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.

Over-specification is the easier of the two to handle, we simply select conflicting genes on a *first-come-first-serve* precedence rule. More often than not, a full gene complement is required in order to evaluate the objective function, hence the method for tackling under-specification if important. Under-specification is handled by making a simplifying assumption about the structure of the fitness function. Templates are used to fill in the unnamed genes with a locally optimal structure. However, since information about locally optimal solution is not usually known, a level-wise mGA is used. In level one, the initial population is created to comprise of all possible string length to a problem, and a random fixed template is used to handle under-specification. In successive levels, the local optimal solution to the previous level is used as the template for the current level.

B. mGA Operators

To handle strings of variable length, the standard *crossover* operator is no longer suitable. Instead it is replaced by two new operators called *cut* and *splice*, Fig. 2. The *cut* operator splits a string at a randomly chosen position with *cutting probability* $P_c=(\lambda-1)P_K$, where P_K is the gene-wise cutting probability and λ is the string length. The *splice* operator concatenates two strings in a randomly chosen order with a fixed probability P_s. The difference between these operators and the crossover operator is that for crossover, the crossover point has to be at the same position for both parents.

Fig. 3 illustrates the pseudo code for messy genetic algorithms. The selection mechanism is as in regular GA but executed in primordial and juxtapositional phases. During the primordial phase, the population is first initialised to contain the all possible building blocks 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. Also, during this phase, the population size is reduced by halving the number of individuals at specified intervals. The juxtapositional phase follows the primordial phase, and here the GA invokes the cut, splice and the other GA operators.

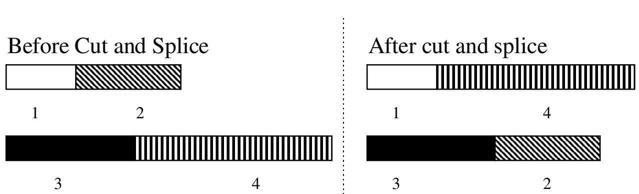

Fig. 2: A messy Cut and Splice operation.

IV. STRUCTURE OPTIMISATION AND LEARNING

The success of an NFC solving any given problem critically depends on the architecture of the network. In addition to influencing the quality and performance of the controller, an inappropriate choice of architecture can lead to slow or no convergence. NFC are typically specified in terms of the topology, functions computed by the neurons and the connection weights.

We start by initialising the NFC, that is by identifying the inputs and outputs; defining an initial number of neurons per layer; types of activation functions (Gaussian, triangular, etc.) of each neuron and the links between adjacent layers. The learning algorithm is based

```
Void mGA
{
        template = zeros;
        for (level=1;level<max_level;level++)
        {
                eval(template,population);
                while (primordial_phase)
                {
                        select(population);
                        reduce_pop(population);
                }
                while (juxtaposition_phase)
                {
                        select(population);
                        cut(population);
                        splice(population);
                        mutate(population);
                }
                template = best(population);
        }
}
```

Figure 3. Messy GA Pseudo Code

on an adaptation of the backpropagation method which mimic fuzzy inferencing and defuzzification. The parameters to learn are the weights which relate to the shape of the activation (or membership) functions. In our case these are the centres and widths of the fuzzy sets. On the first run we take our guestimated NFC and test it against an error minimising function. If it is found that the NFC structure needs adjustments then the process re-enters the loop.

The relevant parts of the network requiring optimisation are layers 1,2 and 4 as only these influence the action of the controller. The other parts on the network are kept constant. Each gene in the mGA is a set of numbers which indicates the I/O index, the neuron of the adjacent layer it connects to and the type of activation of the neuron. Initially we assume a skew-symmetric rule base but no knowledge of the rule base or of the number of membership functions per input/output variable. However we do know the numbers of input and output variables. Consider a coded string as in Fig. 4. Assuming there are 2 inputs and one output, then the string **((1,1,2) (3,2,2) (3,0,1) (2,3,1))** would be interpreted as:

Input 1 *connects to the 1st neuron of layer 2 which has activation of type 2.*

Input 2 *connects to the 3rd neuron of layer 2 which has activation of type 1.*

Output 1 *connects to the 2nd neuron of layer 4 which has activation of type 2.*

This is an overspecified gene because reference to the output is made twice and we handle this as before on a *first-come-first-served* rule. The drawback with coding is that occasionally we could end up with cases where there are no premise or consequence parts (under-specification). In such cases we discard the corresponding connection. The initial template is a mapping of a manually tuned rule base.

The activation functions correspond to the shape of the membership functions which we limit to the two most commonly used, viz. triangular and Gaussian simply for

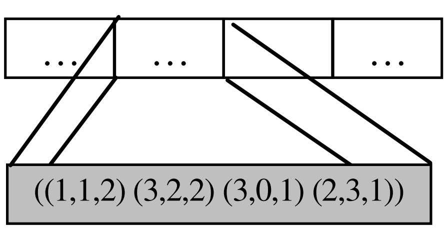

Fig 4. Coding of fuzzy rule base: a sample gene

the facts stated earlier. In this work, a Gaussian type activation was set as *type 1* and triangular activation was set to *type 2*. After the topology of the network is determined the weights are fine-tuned using the learning algorithm.

V. A SIMPLE EXAMPLE

There are many practical engineering robot-arm like applications, such as ballistics, cranes, space shuttle arm, which depend on precision, stability and flexibility. A good prototype is the highly non-linear inverted pendulum on a cart model. The problem is to control the motion of the cart along a horizontal line so that the pole will not fall down and eventually stand vertical, Fig. 5.

The overall control objective or the composite cost function to minimise is described by:

$$\sum_{i=1}^{4 \text{ initial conditions}} \sum_{j=1}^{250 \text{ epochs}} |T_s| + |f| \qquad (1)$$

where T_s is the settling time (defined as the time by which the pendulum must remain stable to $\pm 3^{o}$ to the vertical) and set at 5 seconds, and f is a penalty function defined as the amount the pendulum is away from the vertical. The four initial conditions were used to generalise the solution.

A symmetrical fuzzy rule-base with five fuzzy sets per input/output variable was initially generated for starting the mGA. Fig. 6 shows the resulting structurally optimised neurofuzzy network. Fig 7 illustrates the response of the pendulum for various initial conditions outside the ones used for optimisation for the best final solution. The simulation was also carried out using regular GA and the responses for both the regular GA and mGA optimisation are superimposed to present a comparison. We can see that with the mGA optimisation the pendulum settles to the vertical position very quickly and that the controller robustly offers zero

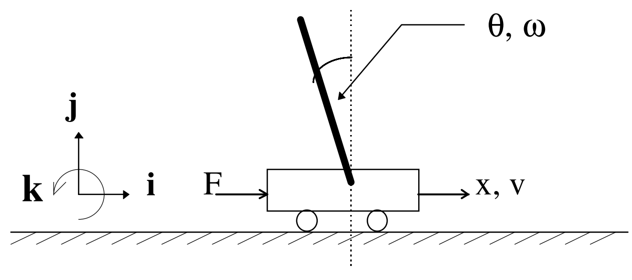

Fig 5. Schematic of the cart-pole model.

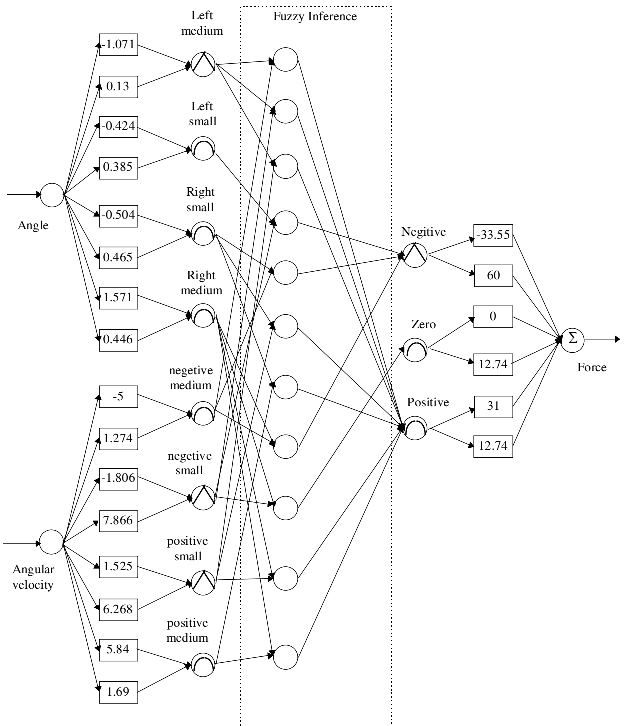

Fig 6. Optimised Neurofuzzy Network

steady state errors even for critical initial conditions. However, the regular GA does not reach vertical position, although it does settle down to within the specified error margin in a shorter time.

VI. CONCLUSION

A new and efficient method of globally and structurally optimising neurofuzzy controllers using messy genetic algorithms has been presented. Structural optimisation is regarded important before ANN based on-line local learning is attempted or switched into. The test results show that the flexible structure of this method provides a means of obtaining a more accurate neurofuzzy controller. Its ability in structural optimisation releases the potential of the nonlinear proportional plus derivative FLC in dealing with steady state errors without an integral term.

An interesting test would be to compare the flexibility of mGAs with other flexible methods such as the genetic programming (GP) [9]. It is expected that a mGA is to yield more logic and faster combinations than GP.

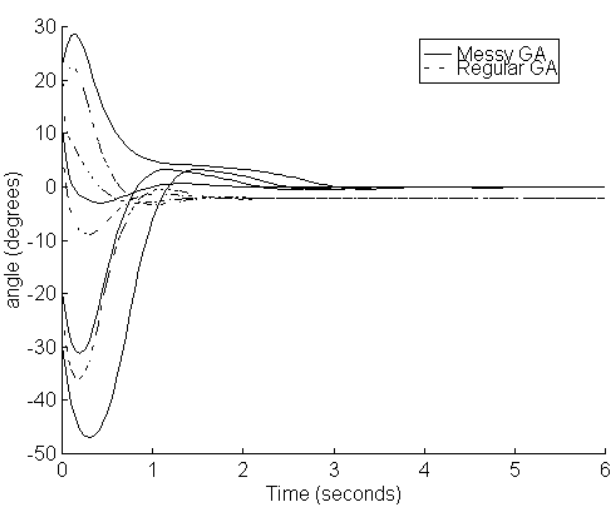

Figure 7. Response of inverted pendulum to various initial conditions.

ACKNOWLEDGEMENT

The authors wish to acknowledge the Engineering & Physical Sciences Research Council of the UK for assisting in the funding for this work.

REFERENCES

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

[2] E.H. Mamdani, "Applications of Fuzzy Algorithms for Simple Dynamic Plant", *Proceedings of the IEEE*, 122(12), 1974, pp. 1585-1588.

[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, 1996, pp. 129-151.

[4] H. Takagi, and M. Lee, "Integrating design stages of fuzzy systems using genetic algorithms". *Proceedings of the Second IEEE International Conference on fuzzy Systems,* 1993, pp. 612-617.

[5] D.E. Goldberg, *et al*. "Messy genetic algorithms: motivation, analysis, and first results." *Complex Systems*, 3, 1989, pp. 493-530.

[6] D.E. Goldberg, *et al*. "Messy genetic Algorithms Revisited: Studies in Mixed Size and Scale", *Complex Systems,* 4, 1990, pp. 415-444.

[7] C. Karr, "Genetic Algorithms for fuzzy controllers", *AI Expert*, 2, 1991, pp. 27-33.

[8] K. Kropp. "Optimization of fuzzy logic controller inference rules using genetic algorithms". *Proceedings of the EUFIT'93*, Aachem, 1993, pp. 1090-1096.

[9] G.J. Gray, Y. Li, D.J. Murray-Smith and K.C. Sharman, "Structural System Identification Using Genetic Programming and a Block Diagram Oriented Simulation Tool", *Electronics Letters*, 32(15), 18 July 1996.