Modifications of Genetic Algorithms for Designing and Optimizing Fuzzy Controllers

J. Kinzel F. Klawonn R. Kruse
Department of Computer Science
Technical University of Braunschweig
D-38106 Braunschweig, Germany
Tel. +49.531.391.3293, Fax +49.531.391.5936, Email klawonn@ibr.cs.tu-bs.de

***Abstract*— This paper investigates the possibilities for applications of genetic algorithms to tuning and optimizing fuzzy controllers, or even to generate fuzzy controllers automatically. There are various ad–hoc approaches to use genetic algorithms for the design of fuzzy controllers, which already indicated good results. However, there is a need for systematic techniques that take the properties of fuzzy controllers and genetic algorithm into account in order to obtain fast convergence and to be able to tackle more complex control problems.**

I. Introduction

Fuzzy control (for an overview see for example [11, 12]) is a control strategy that is not based on a mathematical description of the process to be controlled, but intends to model the behaviour of a human operator who would (theoretically) be able to control the process. The expert's knowledge is specified in terms of linguistic control rules in which expressions like *negative big*, *positive small*, etc. appear. These linguistic expressions are associated with fuzzy sets. In [7, 8] it was shown that fuzzy control can be interpreted as an interpolation process in which the rule base corresponds to the specification of a partial control function. Each linguistic expression can be associated with a crisp value and the respective fuzzy set represents this value taking some imprecision into account.

Although the idea of fuzzy control is very appealing, the design of an appropriate rule base and the specification of the fuzzy sets are very tedious and difficult tasks that are necessary in order to obtain a near optimal control or even a stable control at all. Therefore, the need of at least a partial automatization of these tasks is considered as important and a large number of approaches involving neural networks was proposed (see for instance [1, 9, 14, 16]. Recently genetic algorithms are also considered as a possible solution to the problem of the design and optimization of fuzzy controllers (for an overview see section 4).

The paper is organized as follows. In section 2 we discuss the requirements for a genetic algorithm that are imposed by the fuzzy control concept. Section 3 describes our realization of the ideas developed in section 2. Other approaches are considered in section 4.

II. Pragmatics for Genetic Algorithms from the Viewpoint of Fuzzy Control

The main parameters of a fuzzy controller whose design and tuning are feasible for genetic algorithms are the rule base and the fuzzy partitions. We will not consider the choice of operators like t–norms here.

Fine tuning of a fuzzy controller can only be done by careful adjustments of the fuzzy sets in the fuzzy partitions, but not by a change of the rule base. Therefore, it is reasonable to start with straight forward simple fuzzy partitions and to learn first the rule base and after that carry out changes in the fuzzy sets for fine tuning. A simultaneous learning of the rule base and the fuzzy partitions is possible in principle, but since a bad rule base results in drastic changes for the fuzzy sets, it is better to divide these tasks.

Binary versus Non–Binary Codings

The rule base as well as the fuzzy partitions are generally not given in a binary representation. Although binary codings are better suited for hyperplane sampling [18], for non–binary codings more adequate mutation operators can be defined and the destroying effects of crossover are reduced. To see this consider for example a rule base for a fuzzy controller in the form of a table with 7×7 entries, where for each entry seven possible values (conclusions) are possible. A possible non–binary coding would use $g = 49$ genes, having seven possible alleles for each of them. A binary coding could use $b \times g$ genes where each group of b genes encodes one entry in the rule base. In our case we would have $b = 3$. Building blocks in the binary representation will in general have all four genes of one group as don't care symbols or all four as determined alleles. Assume that we have a schema in the non–binary representation of defining length ℓ. The corresponding binary representation then has a defining length of $b \cdot \ell$. The probability that the application of crossover will destroy this schema can be (over)estimated in the usual way, leading to $\ell/(g-1)$ in the non–binary case, whereas we have $(b \cdot \ell + b - 1)/(b \cdot g - 1)$ for the binary representation. It is easy to check that the latter expression is greater than the first one.

The Building Block Hypothesis

Fuzzy controllers are a very good example where the building block hypothesis is satisfied in a natural way. Since fuzzy control can be seen as an interpolation technique, mostly neighbouring fuzzy sets in the fuzzy partitions have a strong combined effect on the resulting control function. Fuzzy sets that are 'far away' from each other can be treated more or less independently. The linear sequence of fuzzy sets in a fuzzy partition admits an easy coding in the form of a string. Nevertheless, the coding has to be chosen carefully in order to maintain the semantics of the fuzzy sets (positive big should not be on the negative side) and to enable the application of suitable mutation and crossover operators.

The situation for the rule base is a little bit different. We have the same phenomenon that only rules that are neighbours in the rule table do interfere. But the rule table is planar structure whose neighbourhood properties are destroyed by forcing it into a linear string (chromosome). Taking a closer look at the crossover operator, especially for two–point crossover, it is obvious that crossover exchanges a linear subpiece of two linear or ring structures. Thus two–point crossover maintains the neighbourhood structure of the ring, except for those two points where the linear subpiece is cut out [2]. This idea can be generalized to planar or other higher dimensional structures, i.e. we understand a chromosome as an n–dimensional array and crossover exchanges subarrays. For such crossover operators a similar schema theorem can be derived where the defining length of a schema has to be replaced by a notion like the defining 'radius' (of the subarray of determinate genes).

III. Modifications for the Genetic Algorithm

Developing a fuzzy controller is divided in two steps. First an expert has to define a rule base using linguistic variables. After that he defines fuzzy partitions to translate his linguistic rule base into a control function. In our approach genetic algorithms are used to ease the experts work of generating an appropriate rule base and the corresponding fuzzy sets.

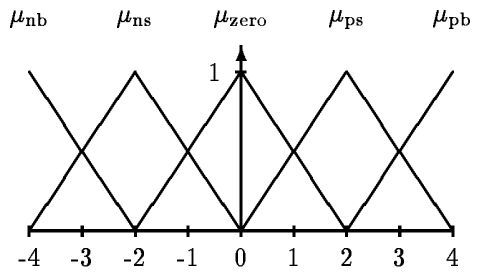


Figure 1: A homogeneous fuzzy partition.

Both tasks are aggregated in one module in the environment of a fuzzy developing system that can make use of a predefined controller, provided by the expert, and generates an optimized solution according to the given problem. This task is solved in three steps:

- find a good initialization for the rule base and the fuzzy sets if no controller is provided by the expert
- generate a rule base which is able to handle the given problem
- tune the fuzzy sets to optimize the performance

Finding a Good Initialization

Genetic algorithms in other approaches usually initialize the rule base randomly, incorporating no knowledge about the control problem. In most cases this rule base differs strongly from the best rule base obtained by optimization. Using implicit knowledge it may be possible to find a better initial rule base, using standard rules like

1. if the antecedents are zero, the consequents should be zero, too,
2. if the antecedents of two rules are similar, the consequences of these rules should also be similar.

A rule base can be generated by induction of these rules. Start with step (i) to generate the conclusion for the rule, where all antecedents are zero. Then proceed with step (ii) for all other conclusions.

The initial fuzzy partition of each domain should be homogeneous. A homogeneous fuzzy partition is illustrated in figure 1.

Generation of a Good Rule Base

To generate a useful rule base, we use a genetic algorithm which adapts the initial rule base according to the given problem. In a first step the population is generated by applying the mutation operator on all genes of the initial rule base. After calculating the fitness of the population, the genetic operations selection, crossover and mutation are used to generate the next population. This is done until a good rule base is found.

Coding of the Rule Base

An appropriate coding of the optimization problem is of significant importance for the performance of a genetic algorithm. The coding should be chosen in accordance to Holland's schema theorem [4, 3]. We will use the following coding which differs from ordinary codings, because it does not use a string of genes but an $n_1 \times \ldots \times n_d$ matrix where n_i is the number of fuzzy sets in domain i. Each element of this matrix contains one fuzzy set of the output domain. This coding is analogous to a representation of the rule base in the form of a table. An example of a coding of a rule base for the cart pole problem is shown below.

	nb	nm	ze	pm	pb
nb	nm	nb	ze	nm	pm
nm	nb	nm	nm	ze	nm
ze	nb	nm	ze	pm	pb
pm	nm	ze	pm	pm	pb
pb	pb	pm	pb	pb	pm

Example of a coded rule base

For example the gene (nm,ze) represents the following rule:

IF φ is nm AND $\dot{\varphi}$ is ze THEN F is nm

Crossover on Rule Bases

According to the two–point crossover operator on gene strings we use a point–radius operator, which exchanges areas of genes between two chromosomes. Both crossover point and radius are chosen randomly.

The following example shows how this crossover operator works. Crossover is exchanges the areas with capital letters as entries. This crossover operator was motivated in the previous section.

nm	nb	ze	nm	pm
nb	NM	nm	ze	nm
NB	NM	ZE	pm	pb
nm	ZE	pm	pm	pb
pb	pm	pb	pb	pm

pb	nm	pb	pm	mm
ze	ZE	nm	nb	pb
PM	NB	ZE	pm	pb
pb	PM	pm	pm	nb
pm	pm	pb	nb	nm

$$\longrightarrow$$

nm	nb	ze	nm	pm
nb	ZE	nm	ze	nm
PM	NB	ZE	pm	pb
nm	PM	pm	pm	pb
pb	pm	pb	pb	pm

pb	nm	pb	pm	mm
ze	NM	nm	nb	pb
NB	NM	ZE	pm	pb
pb	ZE	pm	pm	nb
pm	pm	pb	nb	nm

Mutation of Rule Bases

Mutation is used to change a gene randomly without any respect to the fitness of the chromosome. In our approach one fuzzy set is replaced by a randomly chosen but similar fuzzy set. This means that the fuzzy set "ze" could be mutated either to "nm" or "pm".

Tuning the Fuzzy Sets

Having found a rule base which is able to solve the problem to a certain degree, we can now tune the according fuzzy sets to obtain better performance of the controller. Here the problem is to find a good coding of the fuzzy sets. Most approaches use a bit string coding of characteristic parameters of the fuzzy sets. For example the centerpoint and width of a triangular fuzzy set are coded as binary representations in bit string form. The problems of this coding were discussed above.

We think that this representation is too abstract taking into account the idea that crossover should exchange *good features* between two chromosomes. In our case *good features* are appropriate coverings of a fuzzy partition. Evidently crossover means to exchange parts of the fuzzy partition of two chromosomes. According to this idea we propose the following representation.

Coding of Fuzzy Sets

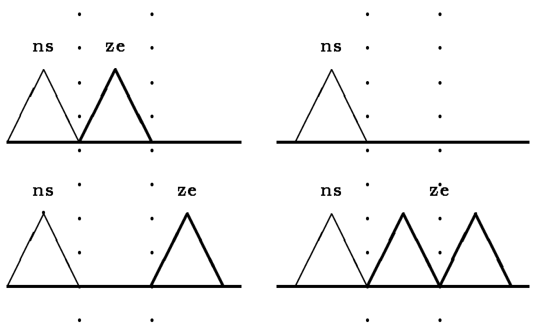
Each domain is represented by a string of genes. Each gene represents the membership values of the fuzzy sets of domain d at a certain x-value. This is how a fuzzy partition is described by discrete membership values.

Suppose a domain d is limited by its left boundary l and its right boundary r, and it is partitioned in n_d fuzzy sets. Its representation is given by:

$$\underbrace{\cdots \overbrace{\begin{pmatrix} \mu_{1_d}(l) \\ \vdots \\ \mu_{n_d}(l) \end{pmatrix}}^{\text{gen}} \cdots \begin{pmatrix} \mu_{1_d}(r) \\ \vdots \\ \mu_{n_d}(r) \end{pmatrix}}_{\text{coding of domain } d} \cdots$$

Crossover on Fuzzy Sets

We use a two–point crossover operator which exchanges the ranges of two given chromosomes. So just ranges of the represented partitions are exchanged. The following examples show how crossover works, and which problem results of this representation.



In most cases we have the undesired side–effect to obtain non convex fuzzy sets. Then we have to use a repair-algorithm to make these sets convex.

Secondly fuzzy sets may vanish. In this case the resulting controller is no longer able to solve the given problem, and the corresponding chromosome either will not be selected in the next generation or the fuzzy set is generated again by mutation in the next step.

Mutation of Fuzzy Sets

Mutation of fuzzy sets is carried out by randomly choosing a membership value $\mu_i(x)$ in a chromosome and changing it to a value in $[0, 1]$. Non convex fuzzy sets will be made convex as said above.

Fitness Function

The fitness function used in a genetic algorithm depends on the problem the controller has to solve. Therefore it is impossible to specify a fitness function in general. But we can extract classes of problems which can be described by a common function. For example the class of problems where a parameter $x(t)$ shall adopt a special value c, we can use the *time–weighted error*

$$f = \sum_{t=start}^{maxtime} t(x(t) - c)^2$$

as it is stated in several articles [10, 6].

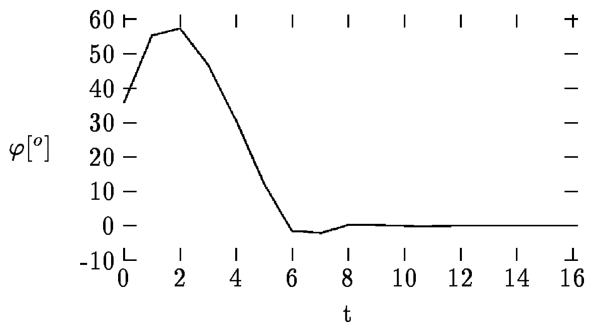
The problem is to select appropriate initial conditions to test a controller's performance. Some authors suggest to select a condition out of the entire parameter space randomly [13, 10]. This includes very extreme conditions for a controller that is not able to manage even easy problems. So we suggest to use easy problems at the beginning and aggravate them when there is a certain amount of chromosomes which manages the easier problems.

Results

We used the cart pole problem to test our approach. The population size was set to 200. The resulting rule base after 33 generations was

	nb	nm	ze	pm	pb
nb	nm	nb	nb	nm	pm
nm	nb	nm	nm	ze	nm
ze	nb	nm	ze	pm	pb
pm	nm	pm	pm	pm	pb
pb	pb	pm	pb	pb	pb

The resulting performance with initial conditions $\varphi = 40.0^o$ and $\dot{\varphi} = -2.0$ is shown in the following diagram:



The tuned fuzzy sets were not much different from the initial ones. This is, because the optimized rule base seems to be very good, and because the cart pole problem is a relative artificial and "easy" problem.

IV. Other Approaches

In this section we shortly review other approaches and discuss the differences to ours.

The Approach of C. Karr

C. Karr [6, 5] uses genetic algorithms to alter just the shape of the fuzzy sets used in a given rule base. Each parameter of a fuzzy set (left–, middle–, and right point) is coded as a seven–bit binary number. The parameters of all sets are concatenated to a bit-string used for the genetic algorithm. The fitness is achieved by calculating the square error of 4 initial conditions chosen randomly from the input space.

This approach shows that even a genetic algorithm with a "simple" coding may result in good solutions. This is because genetic algorithms are robust algorithms in optimizing.

The Approach of Takagi & Lee

H. Takagi and M. Lee [13] use a genetic algorithm to optimize the rule base (including number of rules and fuzzy sets per domain) and the shape of the fuzzy sets of a Takagi–Sugeno controller (TSC) [17]. A rule of a TSC has the form:

IF x_1 is A AND x_2 is B THEN $y = \omega_1 x_1 + \omega_2 x_2 + \omega_3$

ω_i are called *rule–consequent parameters.*

A triangular fuzzy set is described by three parameters. The distance from the leftmost point to the center (leftbase). The distance from the rightmost point to the center (rightbase) and the distance from the center to the previous fuzzy set (center).

The fuzzy sets are coded in the form of *membership function chromosomes* (MFC), e.g.:

leftbase	center	rightbase
10010011	10011000	11101001

Note that leftbase, rightbase, and center are all positive!

The *rule-consequent parameters chromosome* (RPC) is coded analog for the parameters ω_i.

The rule base is coded in a bit string as follows.

domain 1	domain 2	rule consequents
$\mathrm{MFC}_{1\ldots n}$	$\mathrm{MFC}_{1\ldots m}$	$\mathrm{RPC}_{1\ldots nm}$

Crossover and mutation operators are bit manipulating. After decoding the chromosomes, which do not satisfy special constraints are deleted and so the number of rules is decreased.

The fitness function favours chromosomes which erect the pendel in a short time and those which keep the pendel balanced for a longer time. Furthermore it punishes chromosomes with a lot of rules.

The problem aroused by operating on bit–string chromosomes may lead to undesired solutions as discussed above. Further problems may appear, when optimizing all parameters simultaneously, because they influence each other. We think that it is better to keep some parameters fixed and optimize the others in a second step.

An interesting feature of this approach is the ability to decrease the number of rules. We think that many rules do not significantly decrease a controllers speed, but it may be worth to show the expert which rules are really necessary.

The coding of positive distances avoids the problem that fuzzy sets may "overtake" each other. (That means, a fuzzy set representing the concept "nm" may be found right of the set "ze"). This will keep the semantics of the fuzzy sets. To avoid this problem we propose to tune just the antecedent fuzzy sets. If an overtaking took place, it indicates that the fuzzy sets have changed their roles in the rule base or – in other words – the rule base was incorrect. Assuming that we obtained a correct rule base in a first optimization step, overtaking is unlikely.

The Approach of Kropp & Baitinger

K. Kropp and U. Baitinger [10] propose the use of a genetic algorithm to optimize rule bases with another method. They are coding the rule base in form of a bit–string, too. The rule base table is transformed to a string of consequent fuzzy sets, where a fuzzy set is coded by its number as 3–Bit integer. For there are possibly less than eight fuzzy sets, the other bit combinations are mapped to the ones used. So the problem of generating meaningless genes using crossover and mutation is avoided. A disadvantage we see at this point is that not all fuzzy sets appear with the same probability.

The time–weighted error is used as fitness function where 180 input conditions for $\dot{\varphi} = 0$ are tested. By this means the problem of learning just a subset of conditions is avoided. As said above, the same input conditions lead to the same result. Testing all possible conditions may lead to undesired high computation time.

The Approach of Surman, Kanstein & Goser

Surman, Kanstein and Goser [15] propose to add a term to the fitness function that represents the entropy of a rule base. Entropy describes the average number of activated rules. This effect may appear, when the covering of a fuzzy partition is not homogeneous.

V. Conclusions

We have seen, that there are a lot of possibilities to use genetic algorithms for the optimization of fuzzy controllers. They differ from the amount of information which is used to design the genetic algorithm. In our opinion the aim in optimizing a controller is not just a good performance of the obtained controller but also the information an expert can receive from this solution.

References

[1] H.R. Berenji, P. Khedkar, Learning and Tuning Fuzzy Logic Controllers through Reinforcements. IEEE Trans. Neural Networks 3 (1992), 274–740.

[2] K. DeJong, An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD Dissertation. Dept. of Computer and Communication Sciences, University of Michigan (1975).

[3] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison–Wesley, Reading (1989).

[4] J.H. Holland, Adaptation in Natural and Artificial Systems. University of Michigan Press (1975).

[5] C. Karr, Genetic Algorithms for Fuzzy Controllers. AI Expert 2/1991, 27–33.

[6] C. Karr, Fuzzy Control of pH using Genetic Algorithms. IEEE Transactions on Fuzzy Systems 1 (1993), 46–53.

[7] F. Klawonn, R. Kruse, Equality Relations as a Basis for Fuzzy Control. Fuzzy Sets and Systems 54 (1993), 147–156.

[8] F. Klawonn, R. Kruse, Fuzzy Control as Interpolation on the Basis of Equality Relations. Proc. 2nd IEEE International Conference on Fuzzy Systems 1993, IEEE, San Francisco (1993), 1125–1130.

[9] B. Kosko, Neural Networks and Fuzzy Systems. Prentice Hall, Englewood Cliffs (1992).

[10] K. Kropp, Optimization of Fuzzy Logic Controller Inference Rules using a Genetic Algorithm. Proc. EUFIT'93, Aachen (1993), 1090–1096.

[11] C.C. Lee, Fuzzy Logic in Control Systems: Fuzzy Logic Controller, Part I. IEEE Trans. Systems, Man, Cybernetics 20 (1990), 404–418.

[12] C.C. Lee, Fuzzy Logic in Control Systems: Fuzzy Logic Controller, Part II, IEEE Trans. Systems, Man, Cybernetics 20 (1990), 419–435.

[13] M. Lee, H. Takagi, Integrating Design Stages of Fuzzy Systems Using Genetic Algorithms. Proc. 2nd IEEE International Conference on Fuzzy Systems 1993, IEEE, San Francisco (1993), 612–617.

[14] D. Nauck, F. Klawonn, R. Kruse, Combining Neural Networks and Fuzzy Controllers. In: E.-P. Klement, W. Slany (eds.), Fuzzy Logic in Artificial Intelligence, Springer, Berlin (1993).

[15] H. Surmann, A. Kanstein, K. Goser, Self-Organizing and Genetic Algorithms for an Automatic Design of Fuzzy Control and Decision Systems. Proc. EUFIT'93, Aachen (1993), 1097–1104.

[16] H. Takagi, I. Hayashi, NN–Driven Fuzzy Reasoning. Intern. Journ. Approximate Reasoning 5 (1991), 191–212.

[17] T. Takagi, M. Sugeno, Fuzzy Identification of Systems and its Application to Modeling and Control. IEEE Trans. Systems, Man, Cybernetics 15 (1985), 116–132.

[18] D. Whitley, A Genetic Algorithm Tutorial. Technical Report CS–93–103, Dept. of Computer Science, Colorado State University (1993).