Selection of Fuzzy Rules Using a Genetic Algorithm

Jerry J. Cupal and Bogdan M. Wilamowski
Department of Electrical Engineering
University of Wyoming
Laramie, Wyoming, USA

Abstract

A genetic algorithm was used to modify the members of a population of rules for a fuzzy controller. In this work, the example of a truck backing toward a ramp was solved by the controller. An error function was used to rank the members of the population, and the best members became the parents of the next generation. Crossovers were done to further mix the new members, with additional mutations done on individual rules of randomly chosen members. The algorithm proved to converge toward suitable solutions of this problem, starting from 100 randomly chosen sets of fuzzy rules. The technique shows great promise for the automatic synthesis of rules for fuzzy controllers.

Introduction

Fuzzy systems require expert intuition to define the membership functions and the fuzzy rules. Most fuzzy controllers are robust enough to work amazingly well, despite the fact that often this expert intuition is far from optimum. Because it is often necessary and/or desirable to operate a fuzzy controller in an optimum mode, there is an interest in developing techniques for the optimum design of these controllers. Many researchers are attempting to find the optimum solutions for systems whose input-output relations are known [1][2]. For these systems, the fuzzy controller can be trained in a similar way as neural networks [3][4][5] or more advanced methods based on orthogonal least-squares learning algorithms[6]. One possible approach is to use quasi random search techniques through the fuzzy rule tables looking for some optimum performance [7].

In this paper, a genetic search is used to find the best performance. The example of backing up a truck to a ramp is used in this study. The problem is stated the same way as presented in [7] and [8], including the same membership functions. The goal of the genetic search is to find the optimum set of fuzzy rules. In fact, the search algorithm described here can be used to automatize the synthesis process of fuzzy systems.

Problem statement

In the example of backing a truck to ramp, there is no predefined path for each truck location and therefore the optimum steering angle is likewise unknown. Furthermore, a controlled object such as the truck has a certain "inertia", so the correctness of the assumed control variables and not known for some time. In our example, the truck could drive out of the parking lot or crash because of the effect of a wrong set of rules.

The truck is moving back toward the ramp as shown in Figure 1. The motion of the truck can be described by the following set of equations:

$$\begin{aligned} x_{i+1} &- x_i - r\sin(\alpha_i) \\ y_{i+1} &- y_i + r\cos(\alpha_i) \\ \alpha_{i+1} &- \alpha_i + \beta_i \end{aligned} \tag{1}$$

where α is the truck angle, x and y are coordinates of the back of the truck, ß is the steering angle, and r is the incremental driving distance. A fuzzy controller for this problem would have three input variables (α, x and y). However, it can also perform its' function if the variable y is ignored since it is enough to direct truck on the correct track toward the ramp. When truck is directed to the state with $x = 0$ and $\alpha = 0$, then it is only matter of time until it will reach the ramp ($y = 0$). The same approach was used in [8], where y was not used as the input variable for controller. That is, its membership function was not specified.

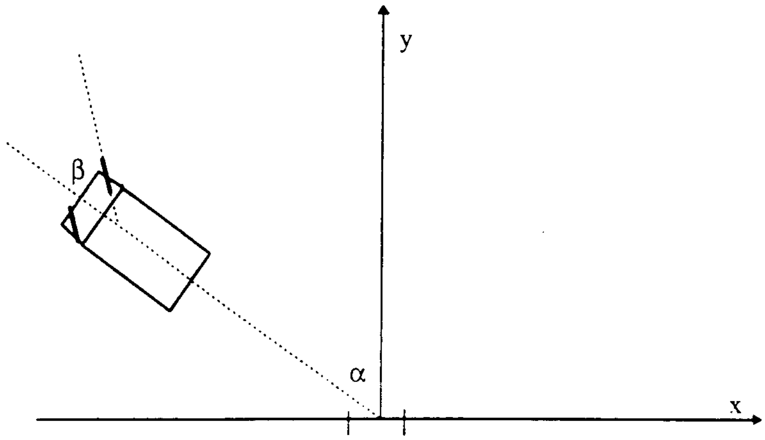

Figure 1. Top view of a truck backing toward a ramp. The ramp is located at the origin.

The membership functions for the inputs and the outputs are the same as in [8]. There are five membership functions for the x input, and seven for both the α input and the β output variables. As a result, the fuzzy rule table consists 35 rules and each rule may has seven different levels. Consequently, there are $35^7 \approx 10^{30}$ possible combinations of rules. It is obvious that a random search is not practical. Even a quasi random search as it was proposed in [7] would require a very long search time. The genetic algorithm seems to be the proper method to obtain a solution to this problem. A genetic search is capable of doing a parallel search of solution space, as opposed to a point-by-point search. By using a population of trial solutions the genetic algorithm can effectively explore many regions of the search space simultaneously, and therefore, it is less sensitive to becoming trapped in a local minima [9].

The genetic algorithm consists of recursively performing the following steps:

1. A given population is tested to rank the members of the population. Ranking is done on certain criterium, usually some error function the measures how well the fuzzy system performs its' task.
2. The population is separated into winners and losers and the losers eliminated. The winners are then reproduced to reestablish the population.
3. The new population is subjected to "Crossovers" where parts of the winners are randomly exchanged.
4. The members of the population are then mutated, where some randomly chosen rules are perturbed slightly.

As the algorithm converges, a population of better and better parents reproduce even better children. The best of these, based upon the error criteria, is then used in the final fuzzy controller.

Experiments

In the experiments done here, a population of 100 members was maintained in each new generation. The members of a given generation were ranked using a cost or error function as follows:

$$Er = \sum_{j=1}^{N_{runs}} \left(x_{end_j}^{\ 2} + 0.1\, \alpha_{end_j}^{\ 2} \right) + \sum_{j=1}^{N_{runs}} D_j \qquad (2)$$

where N_{runs} is the total numbers of trial runs make on a member of the population, x_{end} and α_{end} are the ending positions of the truck, and the driving distance D_j is a measure of how far the truck travels as it approaches the ramp. A particular run was terminated when the position error (the term within the first summation) became less than unity. From Equation 2, it can be seen that the genetic algorithm attempted to align the truck directly in front of the ramp, keeping the driving distance to a minimum.

The cost function was totaled for selected starting points of the truck. For the training process, it was logical to choose values of input variables which correspond to the center values of membership functions. Using the inputs x and α, a training set of 5 * 7 = 35 possible starting points were selected. Because each starting point corresponded to the center of a membership function, initially only one fuzzy rule was applicable. In this way, the rules that started the truck in the correct direction were quickly determined. Each member of the population was ranked using the cost or error function.

After the population of a given generation was ranked, the losers were eliminated and a group of the best individuals were recreated to reestablish a population of 100 members. Several cases were tested; one group that recreated the top 10% of the original population; in another group, 20%. These then became the parents of the next generation.

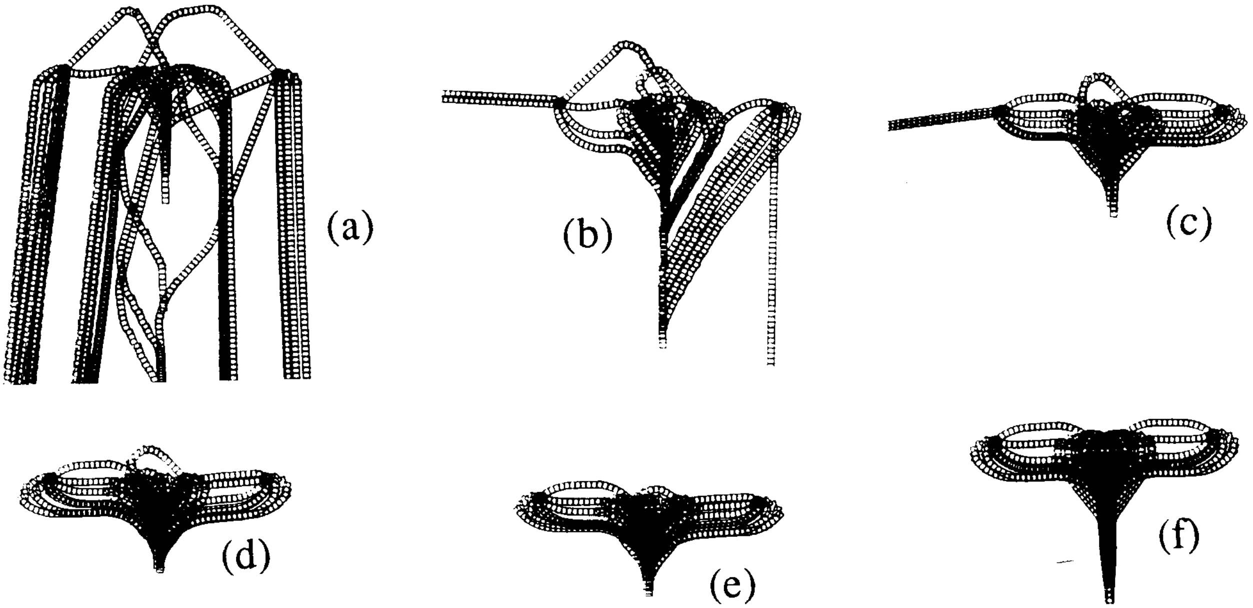

Figure 2. *Paths of the truck as the genetic algorithm converges. The truck starts at 35 different positions. In this test, 10% of a population is retained, with 300 mutations in each generation. Shown are the paths of the best member of the initial population(a) and then after 10(b), 20(c), 30(d) and 40(e) generations. Also shown (in (f)) is the solution from [8].*

This new population was further altered by randomly crossing over (exchanging) one half of the rules between members of the population. A total of 150 such exchanges were made. Then, a given numbers of the individual rules were randomly changed one position in the membership function to add further mutations in the new population. This number varied, in various experiments, from 100 to 3000 to investigate the effect on the convergence of the genetic algorithm.

Many different tests were run. In each, the same original population was passed through the genetic algorithm, with only different number of children retained after each generation, and different number of mutations within each population. The algorithm was allowed to run for 200 generations.

Discussion of results

All of the tests produced sets of fuzzy rules that did guide the truck to the ramp, although some of the paths were certainly not the most direct. A plot of the best set of rules after 0, 10, 20, 30, and 40 generations for the best of these are shown in Figure 2. In this case, 10 members of the population were used in recreation, and 300 individual rules were mutated in each new generation. After about 40 generations, the error did not drop significantly and the algorithm was for all practical purposes converged. For comparison, the paths for the truck when the fuzzy rules are designed by an expert [8] are also shown in Figure 2(f).

Figures 3 and 4 show the error of the best fuzzy sets after each generation. The algorithm seems to converge rather slowly, reaching a steady-state error after 40 generations. The best solutions for all eight tests after the algorithm reached its' 200th generation are shown in Figures 5 and 6. It can be seen that in most cases, the genetic algorithm can find a solution to this fuzzy controller problem.

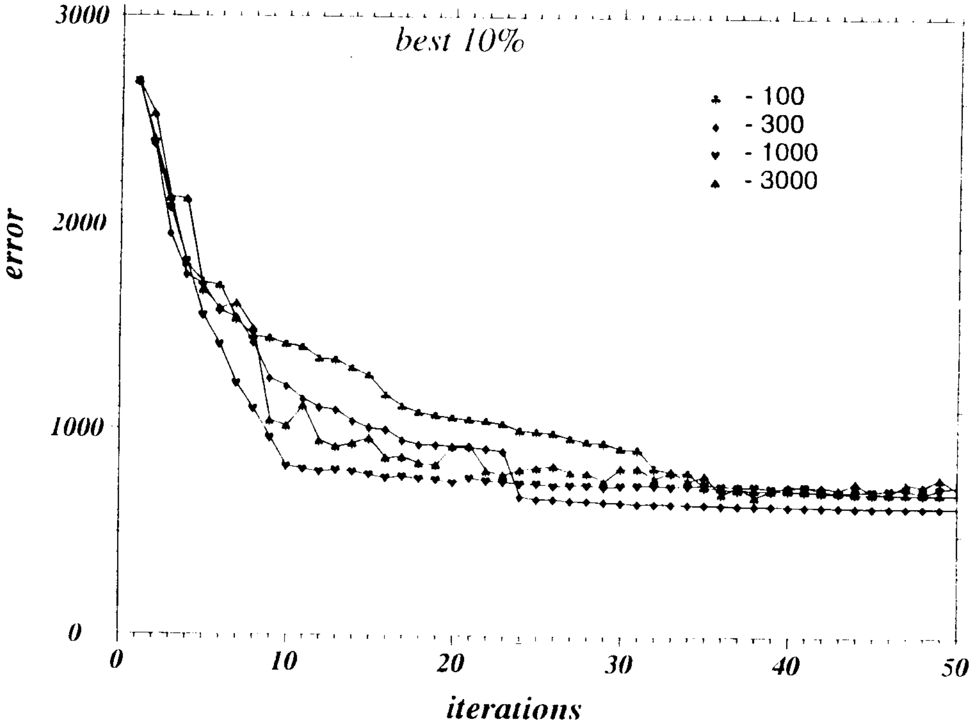

Figure 3. Error function of the population for passes through the genetic algorithm. In these tests, 10% of a given population is retained, with 100, 300, 1000, and 3000 random mutations given to each generation.

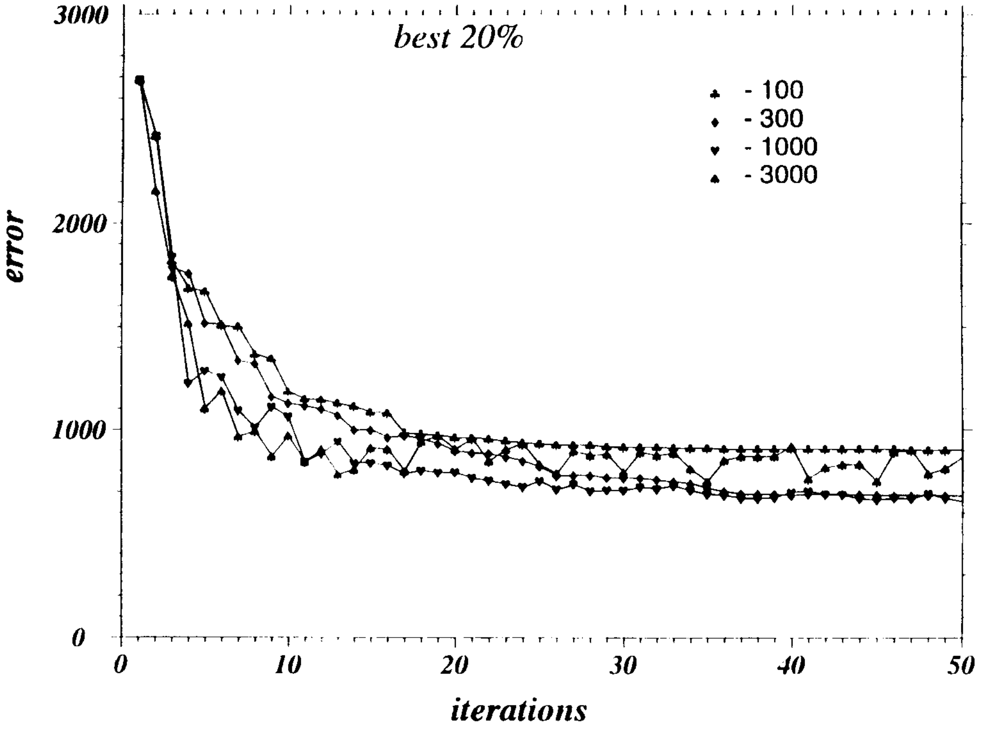

Figure 4. *Error function of the population for passes through the genetic algorithm. In these tests, 20% of a given population is retained, with 100, 300, 1000, and 3000 random mutations given to each generation.*

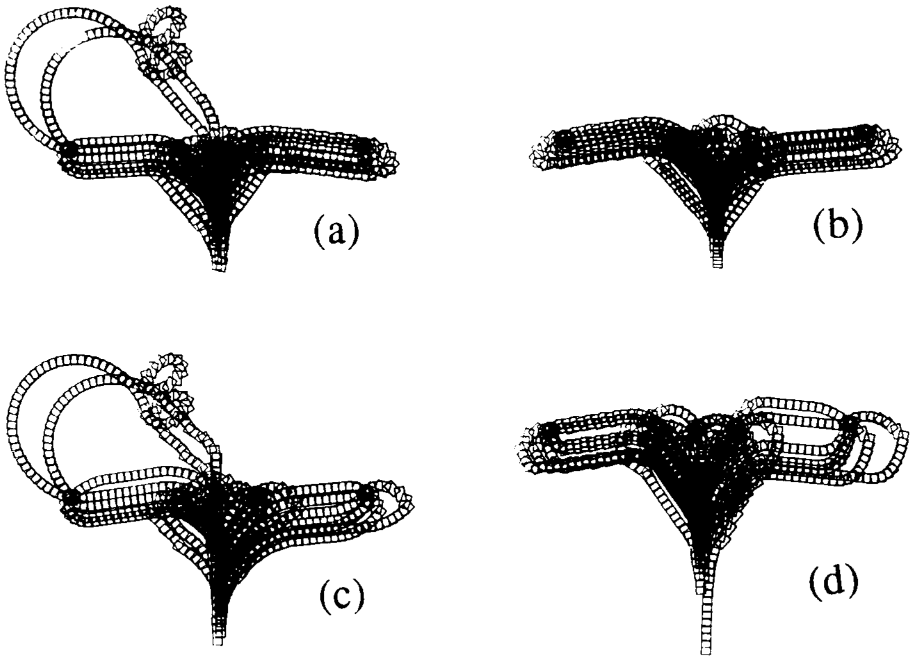

Figure 5. *Paths of the truck for the best member of the 200th generation through the genetic algorithm. In these tests, 10% of a given population is retained, with 100(a), 300(b), 1000(c), and 3000(d) random mutations given to each generation.*

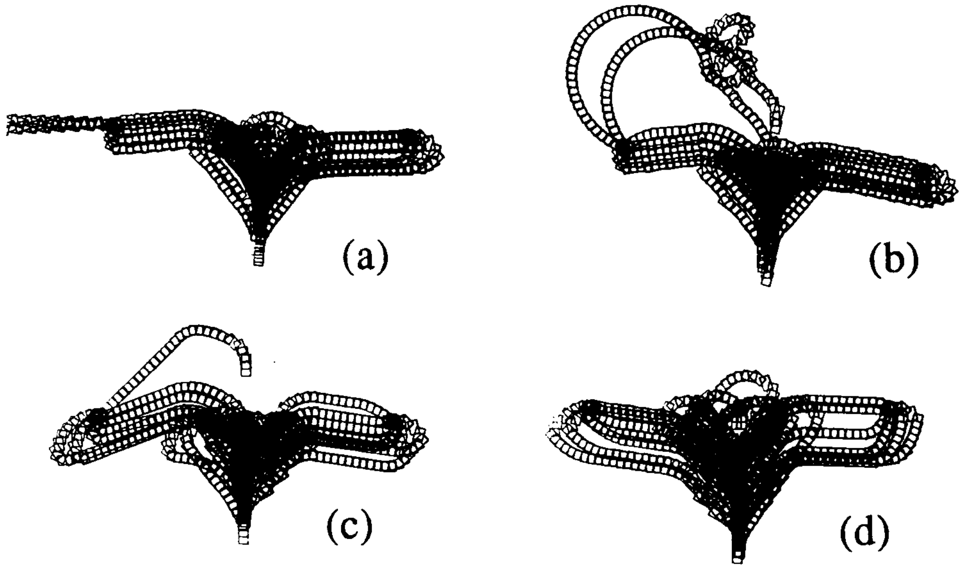

Figure 6. *Paths of the truck for the best member of the 200th generation through the genetic algorithm. In these tests, 20% of a given population is retained, with 100(a), 300(b), 1000(c), and 3000(d) random mutations given to each generation.*

Conclusion

It has been shown that the genetic algorithm can be used successfully for automatic rule finding in fuzzy systems. It was found from experiment that each time a good solution was found, but not the necessarily the best one. Appartently, the algorithm converges to local minimas not far from the global minimum. A disadvantage of the algorithm was that relatively long time was required for convergency.

References

1. J. Bezdek, "Fuzzy models - What are they, and Why," *IEEE Trans. on Fuzzy Systems*, vol. 1, pp. 1-6, Feb. 1993.
2. M. Sugeno, T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling," *IEEE Trans. on Fuzzy Systems*, vol. 1, pp. 7-31, Feb. 1993.
3. S. Horikawa, T. Furuhashi, Y. Usikawa, "On fuzzy modeling using fuzzy neural network with the back-propagation algorithm," *IEEE Trans. on Neural Networks* vol. 3, pp. 801-806, Sept. 1992.
4. D. B. Hertz, Q. Hu, "Fuzzy-neuro controller for backpropagation networks," *Proceedings of WNN 92*, pp 474-478, Auburn, AL, Feb. 10-12, 1992.
5. J. R. Jang, "Self-learning fuzzy controllers based on temporal back propagation," *IEEE Trans. on Neural Networks* vol. 3, pp. 714-723, Sept. 1992.
6. L. X. Wang, J. M. Mendel, "Fuzzy basis function, universal approximation, and orthogonal least-squares learning," *IEEE Trans. on Neural Networks*, vol. 3, pp. 807-814, Sept. 1992.
7. B. M. Wilamowski, R. S. Sandige, "Trainable Fuzzy Controller", presented at *ANNIE'93 - Artificial Neural Networks in Engineering*, St. Louis, Missouri, November 14-17, 1993; also *Intelligent Engineering System Through Artificial Neural Networks*, ASME Pres, vol 3, pp. 561-566.
8. S. Kong, B. Kosko, "Adaptive fuzzy system for backing up a truck-and-trailer," *IEEE Trans. on Neural Networks*, vol. 3, pp. 211-223, March 1992.
9. David Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

Congress On Neural Networks- San Diego

Town & Country Hotel
San Diego, California USA
June 5-9, 1994