An Evolutionary Algorithm for Fuzzy Controller Synthesis and Optimization

Andrea G. B. Tettamanzi*
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
I-20135 Milano, Italy

ABSTRACT

This paper illustrates an evolutionary algorithm for the automatic synthesis and optimization of fuzzy controllers for the Weight Associative Rule Processor (W.A.R.P.) manufactured by SGS-Thomson Microelectronics. The philosophy of this work was to exploit the full potential of such a processor rather than use very simple membership functions and rule structures to reduce the number of degrees of freedom in the search. Details of the algorithm implementation and a discussion of experimental results obtained for the ball-and-beam problem are given.

1 INTRODUCTION

1.1 Evolutionary Algorithms

The name *Evolutionary Algorithm* (EA) encompasses a family of stochastic optimization techniques based on the key concept of *evolution,* such as Genetic Algorithms [2], Evolution Strategies and Evolutionary Programming. A recent work of reference and synthesis in that field is [7].

An EA makes a population of candidate solutions for the problem at hand evolve by iteratively applying a set of stochastic operators, known as *mutation*, *recombination*, *reproduction* and *selection*. Mutation randomly perturbs a candidate solution; recombination decomposes two distinct solutions and then randomly mixes their parts to form a novel solution; reproduction replicates the most successful solutions found in a population; selection purges poor solutions from a population.

The resulting process tends to find globally optimal solutions to the problem much in the same way as natural populations of organisms adapt to their surrounding environment.

1.2 W.A.R.P.

W.A.R.P. is a chip for fuzzy control applications [8] manufactured by SGS-Thomson Microelectronics. Its architecture arises from the need of realizing an integrated structure with high inferencing performance and flexibility. Therefore, W.A.R.P. supports antecedent membership functions with any shape, up to 256 rules, each comprising up to four antecedent clauses and one consequent clause; up to 16 input variables can be handled, described by up to 16 distinct membership functions each, while as many as 128 membership functions can be defined for output variables.

Writing an evolutionary algorithm that is able to exploit the versatility of such a tool to synthesize and optimize fuzzy rule sets for control problems poses an interesting methodological challenge.

1.3 Related work

Other works found in the literature on evolutionary synthesis and optimization of fuzzy controllers ([4, 6, 3] among others), while giving really promising results, tend to use very simple membership functions and rule structures, in order to reduce the number of degrees of freedom in the search process, along with a classical genetic algorithm model. A clear disadvantage of this approach is that much of the power of a processor like W.A.R.P. would remain unexploited.

The spirit of this work points in the opposite direction. An encoding that directly maps to the machine level of W.A.R.P.'s programming language has been defined, and *ad hoc* mutation and crossover operators have been designed to deal with it. For this

*The work described in this paper was carried out while the author was visiting the Computer Science Division of the University of California at Berkeley, supported by SGS-Thomson Microelectronics through Co.Ri.M.Me.

reason this approach could perhaps be classified as a somehow atypical instance of Genetic Programming [5]. Everything, from the number of rules and membership functions to their structure and shape, is left to be discovered by the evolutionary process. In this sense, the only constraints for the evolved controllers are those imposed by the processor's hardware implementation.

A novel model for selection, based on *competition* [9], is employed, which makes it possible to circumvent the difficult and often critical task of defining an appropriate fitness function. Instead, very simple and general rules are used to determine the winner of a stochastic tournament between two controllers: Assuming that failure and success conditions of a simulation are defined, an initial condition is randomly generated and the simulations for both competing controllers are carried out in parallel until one of them fails or succeeds; the competition is then won by the controller that succeeds or by the one that does not fail. Despite its extreme simplicity, this selection model is sufficient for evolution of sensible strategies to take place.

The evolutionary algorithm thus relies on the availability of a simulator for the system to be controlled. It does not try to take advantage of additional knowledge about the problem; however, if available, expert knowledge can be supplied by inserting hand-written rule sets into the initial population.

1.4 The Ball-and-Beam Problem

The ball-and-beam problem has been used as a test case for the evolutionary algorithm presented. Given a beam fixed at its center with a horizontal pivot and free to rotate around it, the task is to maneuver it so as to drive a ball, rolling on it in a groove, from its initial position into a target interval (x_0, x_1), without leaving it fall off either end of the beam.

In the form used by the author, this problem features five input (state) variables (all quantities are measured according to the International System):

- position $-1 \leq x \leq 1$;
- velocity $-10 \leq v \leq 10$;
- acceleration $-g \leq a \leq g$ of the ball with respect to the beam, where $g = 9.81$ is the gravity acceleration;
- inclination $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$;
- angular velocity $-10 \leq \omega \leq 10$ of the beam.

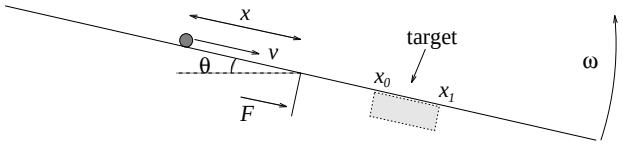


Figure 1: A schematic illustration of the ball-and-beam problem.

The problem has one output (control) variable, the force $-10 \leq F \leq 10$ that has to be applied to the beam.

Figure 1 depicts the system to be controlled. In the version employed, the target interval was defined by $x_0 = 0.5$ and $x_1 = 0.6$. The dynamics of the system is described by the following differential equation:

$$\begin{cases} a(t) = \frac{dv(t)}{dt} = \frac{d^2x(t)}{dt^2} = c[x(t)\omega(t)^2 - g\sin\theta(t)], \\ \frac{d\omega(t)}{dt} = \frac{d^2\theta(t)}{dt^2} = F(t), \end{cases}$$

where $c = 0.7143$.

2 THE ALGORITHM

The overall structure of the algorithm is as follows:

1. Initialize population, either randomly or by reading it from a file;
2. Repeat until the termination condition is met:
   (a) Pick two individuals at random from the population, have them play a competition. The winner will reproduce, while the loser will be replaced by the winner's offspring;
   (b) With probability p_x, pick two individuals at random, have them play a competition and have the winner perform crossover with the winner of the former competition.
   (c) Subject the offspring to random mutation;
   (d) Replace the loser of the first competition with the offspring.
3. Dump the population to a file.

Typologically, this algorithm may be characterized as a steady-state evolutionary algorithm using random binary tournament selection.

The termination condition can be that a number of generations fixed by the user *a priori* has elapsed or it can be triggered by a (fuzzy) predicate on a number of population statistics.

2.1 Encoding

W.A.R.P. programs are encoded in three main chunks: A set of membership functions for the input (or status) variables, a set of symmetric membership functions, represented by means of area-barycenter pairs for the output (or control) variables, and a set of rules.

A single input variable membership function is defined by a list of 32 points; a point is a pair of fixed point numbers fitting into a byte. Output variable membership functions are restricted by W.A.R.P.'s hardware architecture to be symmetrical, thus they can be cheaply described using two parameters: Area and barycenter. A rule is a list of up to four conjoint antecedent clauses (the IF part) and a consequent clause (the THEN part). A clause is represented by a couple of indexes referring to a variable and one of its fuzzy subdomains, i.e. a membership function.

W.A.R.P.'s hardware implementation requires that, for each output variable, two membership functions exist and be used in at least one rule. This requirement is enforced in the algorithm by the convention that the first rules have fixed consequent sides; moreover, the first two membership functions of every output variable coincide respectively with the minimum and maximum of the relevant definition interval, with null area (that is, they represent crisp extreme values for that variable).

2.2 Initialization

The population can be seeded either with handwritten or otherwise already existing W.A.R.P. programs or with new random ones. A new individual is created according to the following algorithm:

1. for each input variable there must be at least one domain; the number of additional domains to be defined for it is determined by sampling from a truncated exponential distribution with mean 3;

2. the shapes of the membership functions for the input variables are determined by random extraction of a centroid C; then two sigmoids having their maximum in C are drawn on both sides to yield an asymmetric bell-shaped curve, whose values in the extrema of the interval $[a, b]$ of the relevant input variable are null:

$$\mu(x) = \begin{cases} \sin \frac{\pi(x-a)}{2(C-a)}, & x < C; \\ \sin \frac{\pi(x-b)}{2(b-C)}, & x \geq C; \end{cases}$$

3. At least two output membership functions have to be present for each output variable; the numer of additional domains for each output variable is determined by sampling from a truncated exponential distribution with mean 3.

4. the barycenters for the output variables are randomly extracted in the range of the relevant variable; the areas are extracted at random such that they correspond to a triangular membership function whose base is entirely contained in the range of the variable;

5. at least two rules have to be present, using at least two different output domains; the number of additional rules in the rule base is determined by sampling from a truncated exponential distribution with mean 6;

6. the rules are generated according to the following algorithm:

   (a) for each input variable, a fair coin is flipped to decide whether to include it in the antecedent part, not exceeding four variables;

   (b) for each selected input variable, a domain is extracted among those defined for it;

   (c) an output variable and a domain for it are extracted for the consequent part of the rule.

2.3 Crossover

The crossover operator implemented preserves the admissibility of W.A.R.P. programs. A new W.A.R.P. program is obtained by combining the pieces of two parent programs. Each rule of the offspring program can be inherited from one of the parent programs with probability 1/2; when inherited, a rule takes with it to the offspring program all the referred domains, along with their membership functions. Other domains can be inherited from the parents even if they are not used in the rule set of the offspring program. These are used as padding material so that the size of the offspring be roughly the average of its parents' sizes.

2.4 Mutation

Mutation too preserves the admissibility of W.A.R.P. programs. It can result in one or more of the following changes, with probability given by the mutation rate, p_m, identical and independent for each component of the genotype:

- a new domain with a random membership function is added to an input variable;

- a domain whose membership function is not used in the rules is removed from an input variable;
- a membership function is perturbed as follows:
  1. a point X in the range of the relevant variable is extracted at random with uniform probability;
  2. a new value $Y \in [0,1]$ for the membership function in X is extracted, again with uniform probability;
  3. the spread σ of the perturbation is extracted from an exponential distribution with mean one quarter of the variable interval;
  4. the old membership function μ is modified into a new one $\bar{\mu}$ such that $\bar{\mu}(X) = Y$,

$$\bar{\mu}(x) = \lambda Y + (1 - \lambda)\mu(x),$$

  where

$$\lambda = \frac{1}{\left(\frac{X-x}{\sigma}\right)^2 + 1}.$$

- a new domain, with random area and barycenter, is added to an output variable;
- a domain whose area and barycenter are not used in the rules is removed from an output variable;
- an area/barycenter pair is perturbed as follows:
  1. a standard deviation σ for the perturbation is extracted from an exponential distribution;
  2. a new barycenter is extracted from a truncated normal distribution with mean the old barycenter and standard deviation σ;
  3. a new area is extracted from a truncated exponential distribution with mean the old area, such that it corresponds to a triangular membership function entirely contained in the range of the relevant output variable;
- a new random rule is added to the rule set; the new rule is generated as follows:
  1. for each input variable, a fair coin is flipped to decide whether to include it in the antecedent part, not exceeding four variables;
  2. for each selected input variable, a domain is extracted among those defined for it;
  3. an output variable and a domain for it are extracted for the consequent part of the rule;
- a rule is removed from the rule set;
- a rule gets a random antecedent clause predicating an input variable not yet used added to it;
- an antecedent clause is removed from a rule;
- the predicate of an antecedent clause is modified by randomly extracting one of the domains defined for the relevant input variable;
- the predicate of the consequent clause of a rule is modified by randomly extracting one of the domains defined for the relevant output variable.

2.5 Selection

Competitive selection [9] is a form of stochastic binary tournament selection. It relies on a stochastic function which, given two competing individuals, chooses the winner according to a probability.

A random initial condition is generated, then the behaviour of both competitors is emulated on two parallel simulations of the object problem. A competitor loses as soon as it violates any constraint of the problem; a competitor wins if one of three cases happens:

- its opponent loses;
- the simulation comes to an end and the controller is in a target condition, while its opponent is not;
- when the simulation time is elapsed the controller is closer to the goal than the opponent.

A tie may result if both competitors fail or succeed at the same time, or if as the simulation time is over their distance from the goal are equivalent; in that case the individual having fewer rules wins. Selective pressure can be adjusted by the use of multiple competition matches.

Such a selection scheme doesn't require the explicit definition of a fitness function. Although probably not the most efficient choice, this should result in a greater generality of the approach.

3 EXPERIMENTAL RESULTS

The algorithm described above was implemented on an HP9000 712/60 workstation, and a number of experimental runs were performed trying several parameter settings. The best results were obtained with a population size not below 64 individuals, mutation rate comprised between 0.001 and 0.1 and crossover rate p_x between 0.1 and 0.2.

The dynamics of the ball-and-beam problem were simulated using a simple Euler method with a step of 0.5 ms; the sampling rate for the controller was set to 20 Hz.

Execution times for a million steps (a step being the generation of a new offspring individual) varied between 24 and 48 hours, depending on the selective pressure.

The complexity of the ball-and-beam system, along with the intentional lack of a performance (fitness) measure, doesn't allow an easy quantitative description of results. Instead, the author deems more significant to give a qualitative description of the controllers synthesized by the algorithm.

Overall, the evolution appears to go through three partially overlapping stages:

1. In the first stage the population learns how to respect the problem constraints, that is, in the case of the ball-and-beam, to keep the ball on the beam without letting it fall off either end.

2. In the subsequent stage, which begins when more or less all individuals in the population are able not to violate any constraint, the population learns how to reach the goal, that is, for the ball and beam, to drive the ball into the designated area of the beam and hold it within as long as possible.

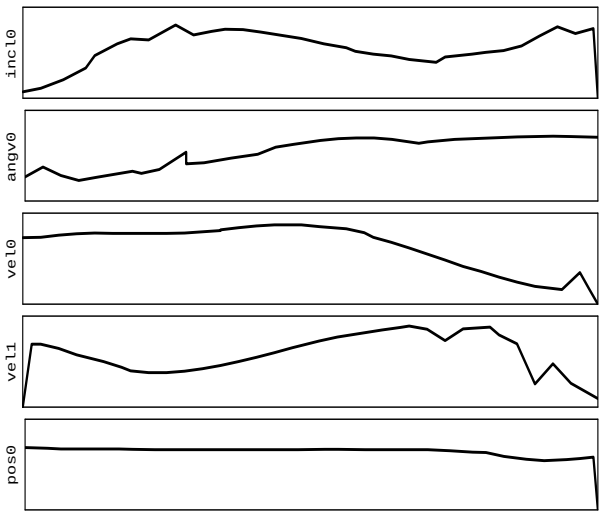
3. The last stage begins when the majority of individuals manage to reach the goal in most cases. At that point what makes the difference in evolutionary terms is the speed at which a controller is able to reach the goal and therefore the population starts optimizing the control strategy with respect to time.

3.1 The solutions

It is interesting to briefly examine what the controllers synthesized by the algorithm look like and how they behave.

In almost all the cases two rules were sufficient to account for the strategy solving the ball-and-beam problem. It appears that the bulk of the information defining a strategy resides in the shapes of the membership functions associated with the input variables, whereas the output membership functions degenerate into crisp numbers, F_- and F_+, placed at the two extrema of the feasible interval. Figure 2 shows a sample controller evolved by the algorithm.

The strategy used by all the evolved controllers can be described as follows:



IF ω is angv_0 AND x is pos_0 AND v is vel_0
THEN F is F_-
IF θ is incl_0 AND v is vel_1 THEN F is F_+

Figure 2: A controller for the ball-and-beam problem evolved by the algorithm. The five membership functions are plotted across the definition interval of the relevant variables

1. Slow the ball down, by tilting the beam in the opposite direction with respect to the motion of the ball, and in proportion with its speed, until the ball sits almost still around the center of the beam.

2. Push the ball toward the target interval with short, gentle jerks in the right direction, immediately balanced by other ones the opposite way. As a result, the ball moves by fits and starts, but at a constant pace, until it enters the target interval.

3. Keep the ball confined within the target interval, by suddenly tilting the beam as the ball approaches either boundary.

In all the cases observed, this last part of the strategy was not completely successful, in that every now and then the ball manages to escape from the target interval for a very short time, before being pushed back into it.

4 SUMMARY AND DISCUSSION

This paper presented an evolutionary algorithm for the automatic synthesis and optimization of fuzzy controllers for the W.A.R.P. fuzzy microprocessor.

The overall structure of the algorithm, as well as the encoding and the *ad hoc* definitions of crossover and mutation were illustrated.

The proposed approach was tested on the ball-and-beam problem, and the risults were qualitatively described.

Discussion The computational effort required to synthesize controllers for the ball-and-beam problem using the proposed approach might look, to say the least, preposterous to a reader with a background in engineering. However, the algorithm, as it is, is designed on purpose not to take any advantage not only of problem-specific expert knowledge that may be available, but also of control engineering common sense. Improvements in these directions are always possible: For instance, one could predetermine the shape and number of membership functions for each variable, or fix a minimum and maximum number of rules, as well as seed the population with rough handcrafted approximations of the controllers sought for. This is not the point of this paper. The point is to show that evolutionary techniques can come up with programmings that make efficient and effective use of all the degrees of freedom provided by a fuzzy microprocessor like W.A.R.P. Besides, even two days of CPU time on a workstation are less expensive than one hour of the time of a control engineer.

Future directions Extensions of this work are possible in two major, orthogonal directions.

First of all the algorithm can be made more specific for certain classes of control problems by encapsulating more control engineering expertise in the representation and in the operators.

On the other hand the algorithm can be refined for a more efficient use of the computing resources. A technique that the author has already used in the past with good results consists in adding a *fuzzy government* module to the algorithm [1], to dynamically tune its parameters and detect the emergence of an optimal solution.

Acknowledgements

The author would like to thank all the people who made this research possible: Rinaldo Poluzzi of SGS-Thomson for help with W.A.R.P. documentation, Roger Jang for providing the dynamics of the ball-and-beam problem, Mike Lee for many stimulating discussions, Lotfi Zadeh for his valuable suggestions and Gianni Degli Antoni for his support.

References

[1] S. Arnone, M. Dell'Orto, and A. Tettamanzi. Toward a fuzzy government of genetic populations. In *Proceedings of the 6th IEEE Conference on Tools with Artificial Intelligence (TAI'94)*, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[2] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[3] C. Z. Janikow. A genetic algorithm for learning fuzzy controllers. In *Proceedings of the ACM Symposium on Applied Computing*, New York, 1994. ACM Press.

[4] C. L. Karr. Design of an adaptive fuzzy logic controller using a genetic algorithm. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann.

[5] J. R. Koza. *Genetic Programming: on the programming of computers by means of natural selection.* The MIT Press, Cambridge, Massachussets, 1993.

[6] M. Lee and H. Takagi. Integrating design stages of fuzzy systems using genetic algorithms. In *Proceedings of the 2nd International Conference on Fuzzy Systems (FUZZ-IEEE'93)*, pages Vol. I, 612–617, 1993.

[7] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1992.

[8] SGS-Thomson Microelectronics. *W.A.R.P. Software Development Tool User's Guide*, version 1.0 edition, 1994.

[9] A. Tettamanzi. A distributed model for selection in evolutionary algorithms. Technical Report 110-94, Dip. di Scienze dell'Informazione – Università degli Studi di Milano, Milano, Italy, 1994.