AN EVOLUTIONARY ALGORITHM FOR FUZZY CONTROLLER SYNTHESIS AND OPTIMIZATION BASED ON SGS-THOMSON'S W.A.R.P. FUZZY PROCESSOR

RINALDO POLUZZI, GIAN GUIDO RIZZOTTO
C.A.S.A. Group, SGS-Thomson Microelectronics, Via Olivetti 2
I-20041 Agrate Brianza, Milano, Italy
E-mail: {rinaldo.poluzzi,gianguido.rizzotto}@st.com

ANDREA G. B. TETTAMANZI
Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39/41
I-20135 Milano, Italy
E-mail: tettaman@dsi.unimi.it

This chapter 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 interest of this work is still speculative, as its underlying philosophy was to try to exploit the full potential of such a processor rather than to use simplified membership functions and rule structures to reduce the number of degrees of freedom in the search for an optimal controller. Other original aspects of the work presented in this chapter include: the use of a competition-based selection scheme that is particularly suited and intuitive for control applications and doesn't require the explicit definition of a fitness function; the resulting definition of a fuzzy fitness; and the use of a "fuzzy government", i.e. a fuzzy rule set in charge of monitoring and controlling the evolutionary process to optimize its performance. Details of a prototypical algorithm implementation and a discussion of experimental results obtained by applying the algorithm to the ball-and-beam problem are given.

Keywords: Evolutionary Algorithms, Fuzzy Logic, Soft Computing, Control.

1 Introduction

The application of fuzzy logic to control problems has become common practice, particularly in Eastern Asia and Europe, and interest has been growing in the U.S. as well. A reason for this success is that it is easier for human experts to express their knowledge about a system in the form of rules that put linguistic variables in relation with one another. After a little tuning, controllers built in such a manner can work as well as any controller obtained following the guidelines of classical control theory, except that much less effort and expertise is needed. In some cases, for particularly complex systems whose dynamics are highly non-linear, applying classical control theory becomes too difficult, and fuzzy control is the only viable resort.

After the first successful applications of fuzzy logic to control problems, speculation arose as to whether it would be possible to develop an automated method to either synthesize the rules of a fuzzy controller from scratch or tune a given set of human-written rules in order to obtain better performance. Evolutionary algorithms have been found to be promising instruments in the optimization of fuzzy controllers.

1.1 *Evolutionary algorithms*

The name *Evolutionary Algorithm* encompasses a family of stochastic optimization techniques based on the key concept of *evolution,* such as Genetic Algorithms,[3] Evolution Strategies [11] and Evolutionary Programming.[6] A recent work of reference and synthesis in that field is Ref. 10.

An evolutionary algorithm 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 environments.

1.2 *Evolutionary synthesis of fuzzy controllers*

The task of using a genetic algorithm or, broadly speaking, an evolutionary algorithm for the design of fuzzy controllers was first undertaken at the beginning of this decade by C. L. Karr [5] and P. Thrift.[14] Similar work has been conducted by Michael Lee and Hideyuki Takagi [8,9] at University of California, Berkeley and by Cezary Janikow [4] at University of Missouri, St. Louis.

A common trait among these approaches is that they use very simple shapes for the membership functions (i.e. triangular or trapezoidal), so that these can be compactly encoded by two or three parameters, and they assume that the number of fuzzy domains for each variable is given and fixed. Furthermore, all possible rules, obtained as a Cartesian product of the input variables, are taken into account and their inclusion in the rule set of a controller is usually marked by a bit in the genetic encoding.

These simplifications greatly reduce the degrees of freedom in the search for an optimal controller while guaranteeing a sufficient amount of generality. The design process, however, is reduced to a parameter optimization problem in the framework of a rigid structure. In addition, the structure and implementation of the controllers is tailored for the convenience of the genetic algorithm.

Although the results obtained thus far are promising and this approach might prove to be entirely satisfactory from the practical point of view, it is not applicable to the case in which a *language* for the design of fuzzy controllers and a hardware architecture for fuzzy inference are given and a fuzzy controller has to be designed upon them. The availability of a dedicated fuzzy control processor, W.A.R.P., prompted the authors to explore an alternative approach to the synthesis and optimization of fuzzy controllers using evolutionary algorithms. This approach could be described as *bottom-up,* in the sense that it starts from an existing fuzzy implementation from which it tailors a design technique rather than defining a suitable implementation given a design technique.

1.3 *The weight associative rule processor*

W.A.R.P. is a chip for fuzzy control applications [12] manufactured by SGS-Thomson Microelectronics. The architecture of the W.A.R.P. chip arose from the need of realizing an integrated structure with high inferencing performance and flexibility. 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.

1.4 *Summary of the approach*

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

A clear disadvantage of standard approaches that use simplified membership functions and rule structures is that much of the power of a processor like W.A.R.P. would remain unexploited.

The work described in this chapter is essentially different than the above-described approaches used in the design of fuzzy controllers, in that an encoding that directly maps to the machine level of W.A.R.P.'s programming language has been defined, and specialized mutation and recombination operators have been designed. This approach could therefore be classified as an atypical instance of Genetic Programming.[6] Every parameter involved in the design, from the number of rules and membership functions to their structure and shape, is discovered by the evolutionary process; the only constraints for the evolved controllers are those imposed by the processor's hardware implementation.

A novel model for selection, based on *competition*,[13] is employed, which makes it possible to circumvent the difficult and often critical task of defining an appropriate fitness function. 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 either the controller that succeeds or the one that does not fail. Despite its extreme simplicity, this selection model is sufficient for the 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.

2 Algorithm Implementation

The overall structure of the algorithm is as follows:

1. Initialize the 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 and devise 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 and devise a competition. The winner will 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 could be either time-dependent (e.g. a predetermined number of generations has elapsed) or triggered by a (fuzzy) predicate on a number of population statistics.

2.1 *Encoding*

W.A.R.P. programs are encoded in three main blocks: a set of membership functions for the input (or status) variables, a set of symmetric membership functions, represented by means of area-center of mass 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 instructed by W.A.R.P.'s hardware architecture to be symmetrical, and thus can be described using just two parameters: area and center of mass. 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 pair 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 hand-written 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. Each input variable must have at least one domain; the number of additional domains 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 with their maximum in C are drawn on both sides to yield an asymmetric bell-shaped curve, whose values in the extremes of the interval $[a, b]$ of the relevant input variable are null:

$$\mu(x) = \begin{cases} \sin \frac{\pi(x-a)}{2(C-a)}, & x < C; \\ \sin \frac{\pi(x-b)}{2(b-C)}, & x \geq C; \end{cases} \tag{1}$$

3. At least two output membership functions have to be present for each output variable; the number of additional domains for each output variable is determined by sampling from a truncated exponential distribution with mean 3.
4. The centers of mass 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;
   (c) an output variable and its domain are extracted for the consequent part of the rule.

2.3 *Recombination*

The recombination 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 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, to increase the size of the offspring so that its size is roughly the average of its parents' sizes.

2.4 *Mutation*

Like recombination, mutation also preserves the admissibility of W.A.R.P. programs. Mutation 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 $\bar{\mu}(X) = Y$,
  $$\bar{\mu}(x) = \lambda Y + (1 - \lambda)\mu(x), \tag{2}$$
  where
  $$\lambda = \frac{1}{\left(\frac{X-x}{\sigma}\right)^2 + 1}. \tag{3}$$
- a new domain, with random area and center of mass, is added to an output variable;
- a domain whose area and center of mass are not used in the rules is removed from an output variable;
- an area/center of mass pair is perturbed as follows:
  1. a standard deviation σ for the perturbation is extracted from an exponential distribution;
  2. a new center of mass is extracted from a truncated normal distribution with mean the old center of mass 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;
3. an output variable and its domain 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[13] 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 behavior 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:

1. Its opponent loses;
2. The simulation comes to an end and the controller is in a target condition, while its opponent is not;
3. The simulation time has elapsed and 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, when the simulation time is over, their distances 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 does not require the explicit definition of a fitness function. Although slightly inefficient, this should result in a greater generality of the approach.

3 The Fuzzy Government

Evolutionary algorithms are relatively easy to implement and, in general, their performance is satisfactory considering the small amount of knowledge they need in order to work; however, typically they would require some sort of human supervision during their use as practical tools.

A remedy for the inconvenience of human supervision is to use a fuzzy knowledge base (or *fuzzy government* [1]) to detect the emergence of a solution, to dynamically tune algorithm parameters, as in Ref. 7 or 2, and to monitor the evolutionary process to avoid undesired behaviors such as premature convergence.

3.1 *The fuzzy fitness*

A fuzzy estimate of the fitness of individuals is given based on the recorded outcomes of competitions. This estimate is based on three quantities: the number c of competitions undergone by an individual, the number w of its wins and the number s of successes. The more competitions an individual has gone through, the less fuzzy its fitness will be.

The membership function of fitness, for a given individual, is defined as:

$$\mu_f(x) = N(a,b)x^a(1-x)^b, \tag{4}$$

where

$$N(a,b) = \frac{(a+b)^{a+b}}{a^a b^b} \tag{5}$$

is a normalization factor and $a = w+s$, $b = c-s$. The mode of the membership function, where its value is one, is

$$\arg\max_{x\in[0,1]} \mu_f(x) = \frac{a}{a+b} = \frac{w+s}{w+c}. \tag{6}$$

An interesting predicate defined on pairs of individuals is their *fitness overlap:*

$$f_\wedge(\gamma,\kappa) = \max_{x\in[0,1]} \min\{\mu_{f(\gamma)}(x), \mu_{f(\kappa)}(x)\}, \tag{7}$$

i.e. the maximum of the intersection of their fitnesses. This gives the degree to which γ and κ have the same fitness.

The fuzzy fitness is employed to avoid carrying out useless competitions: when the fitness overlap of the two opponents is below a certain threshold, the one with the highest mode is awarded the match.

Fuzzy fitnesses of individuals in the population are aggregated to yield statistics such as minimum, average and maximum fitness for use by the fuzzy government, as explained in the next section.

3.2 *Statistics and parameters*

The algorithm has facilities for writing fuzzy rules involving population statistics in their antecedent parts and algorithm parameter values in their consequent parts. At regular intervals, the statistics are computed and an inference is carried out on the rules defined by the user. The algorithm parameters are then updated accordingly and evolution is resumed.

Statistics that can be used are:

- Genotypic diversity D_Γ, that is the probability that two individuals randomly extracted from the population have different genotypes;
- Phenotypic diversity D_Φ, computed on the basis of the average fitness overlap as defined in Eq. (7), yielding the expected degree to which two individuals randomly extracted from the population would have different fitness;
- Maximum fitness f^*, defined as the mode of the fuzzy fitness measure of the best individual in the population. The best individual is the one with the highest fitness mode and the least fitness spread;
- Average fitness $\bar{f}$, defined as the mean of the fitness distribution in the population (the distribution is obtained by summing up the fitness measures for all individuals and normalizing);
- Minimum fitness $f_{\min}$, defined as the mode of the fuzzy fitness measure of the worst individual in the population. The worst individual is the one with the lowest fitness mode and the least fitness spread;
- Fitness range $R = f^* - f_{\min}$;
- Tie rate P_{tie}, defined as the fraction of competitions that end in a tie;
- Success rate P_{succ}, i.e. the fraction of competitions that end because either opponent reached the target;
- Actual mutation rate: this may differ from the mutation rate set through the relevant parameter, due to statistical fluctuations or failures to preserve the feasibility of a solution while attempting to mutate it;
- Time-out rate $P_{\text{t/o}}$, the fraction of competitions that end because neither competitor reached the target and the simulation timed out.

The algorithm parameters that can be acted upon are:

- Mutation rate p_m;
- Crossover rate p_x;
- Selective pressure S;
- Window of success W, i.e. the time a controller must spend in a target state in order to be considered successful;
- Emergence, that is the degree to which a satisfactory solution has been obtained;
- Premature convergence, that is the degree to which evolution is stuck in a local optimum.

The latter two parameters are used to decide whether to stop the algorithm.

Statistics and parameters are in part universal to any evolutionary algorithm and in part specific to a particular application. Therefore it is hard to come up with general rules to control the evolutionary process. These are to be discovered by the implementer by means of experiments and empirical observations.

3.3 *Rules*

All statistics and parameters are normalized in the range $[0, 1]$. Six membership functions are predefined on this domain for use on all variables. These membership functions are of the fuzzy threshold type; triangular or trapezoidal membership functions can be obtained by composition of two or more of them via conjunction and negation.

A fuzzy threshold membership function μ can be defined as follows, for all $x \in [0, 1]$:

$$\mu(x) = \begin{cases} 1, & x \leq \alpha; \\ (\beta - x)/(\beta - \alpha), & \alpha < x < \beta; \\ 0, & x \geq \beta. \end{cases} \tag{8}$$

The values of parameters α and β for the six predefined membership functions are listed in Table 1.

In addition, the modifiers *VERY* and *LITTLE* are defined: *VERY* squares its argument, whereas *LITTLE* returns the square root of its argument. As it is well-known, squaring the membership function has the effect making the borders of a fuzzy set "sharper", whereas taking its square root makes it, so to say, "sloppier".

Table 1: Parameters α and β for the membership functions used by the fuzzy government.

name	α	β
Z	0	0.1
S	0.1	0.25
MS	0.25	0.5
M	0.5	0.75
ML	0.75	0.9
L	0.9	0.1

Figure 1 shows a sample fuzzy government used by the authors to control the algorithm for an application to the ball-and-beam problem described in the next section.

3.4 *Empirical observations*

Since the aim of the fuzzy government is to keep the evolutionary process in a dynamical equilibrium, its overall structure will be based on negative feedback as a guiding principle.

For instance, as a general rule, the mappings between population statistics and algorithm parameters defined by a fuzzy government will inversely relate the mutation rate with diversity (the less diverse the population, the higher the mutation rate).

For the application described, it has been observed that convergence to an optimal controller is foreshadowed by an increase in the number of competitions that time out, i.e. because both opponents are so successful in controlling the systems that they reach the target almost at the same time and neither fails until the whole simulation time has elapsed. Therefore, the time out ratio is in direct relation with emergence.

Several experiments have shown that the population maintained by the algorithm improves at the fastest pace when the success rate is not too high. As a consequence, the time a controller must spend in a target state in order to be considered successful should be linked to the success rate. An increase in the success rate will thus trigger an extension of that time, which in turn will lead to a decrement of the success rate. The rules have to be tuned to attain the most advantageous equilibrium point.

IF P_{tie} is Z THEN P_m is 0.0
IF NOT P_{tie} is Z AND P_{tie} is MS THEN P_m is 0.004
IF NOT P_{tie} is MS AND P_{tie} is M THEN P_m is 0.01
IF NOT P_{tie} is M THEN P_m is 0.1
IF P_{succ} is S THEN W is 0.0
IF NOT P_{succ} is S AND P_{succ} is MS THEN W is 0.01
IF NOT P_{succ} is MS THEN W is 0.02
IF VERY NOT $P_{\mathrm{t/o}}$ is MS THEN Emergence is 0.5
IF NOT $P_{\mathrm{t/o}}$ is VERY M THEN Emergence is 1.0
IF NOT $P_{\mathrm{t/o}}$ is MS THEN W is 0.02

Figure 1: The rule set of a sample fuzzy government.

4 Experiments and 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 without a fuzzy government 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 use of a fuzzy government noticeably improved the performance of the algorithm in terms of time for equal quality of the solutions found.

4.1 *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 free to rotate around a horizontal pivot fixed at its center, the task is to maneuver it so as to drive a ball (rolling in a groove on the beam) from its initial position into a target interval (x_0, x_1) without falling off either end of the beam.

In the form used by the authors, the ball-and-beam problem features five input (state) variables (all quantities are measured according to the International System):

- position $-1 \leq x \leq 1$;
- velocity $-10 \leq v \leq 10$;

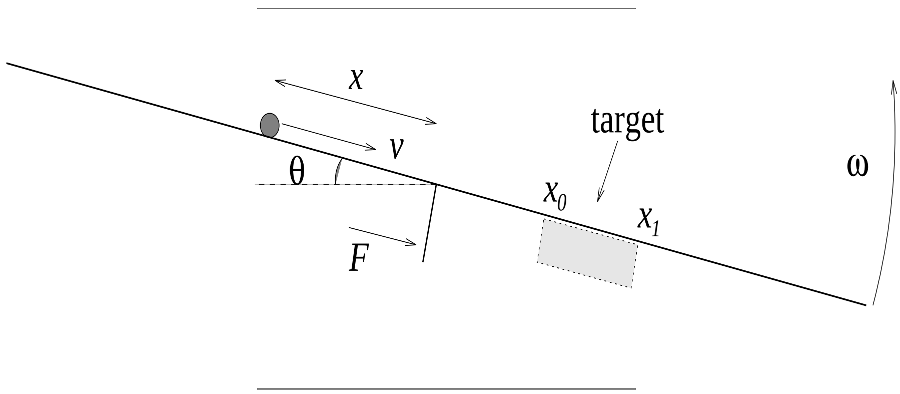


Figure 2: A schematic illustration of the ball-and-beam problem.

- 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.

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

Figure 2 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 are described by the following differential equation:

$$\begin{cases} a(t) = \frac{dv(t)}{dt} = \frac{d^2x(t)}{dt^2} = c[x(t)\omega(t)^2 - g\sin\theta(t)], \\ \frac{d\omega(t)}{dt} = \frac{d^2\theta(t)}{dt^2} = F(t), \end{cases} \tag{9}$$

where $c = 0.7143$.

4.2 *Experiments*

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.

4.3 *Results*

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 authors deem 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 to respect all constraints, 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, therefore triggering the optimization of the control strategy with respect to time.

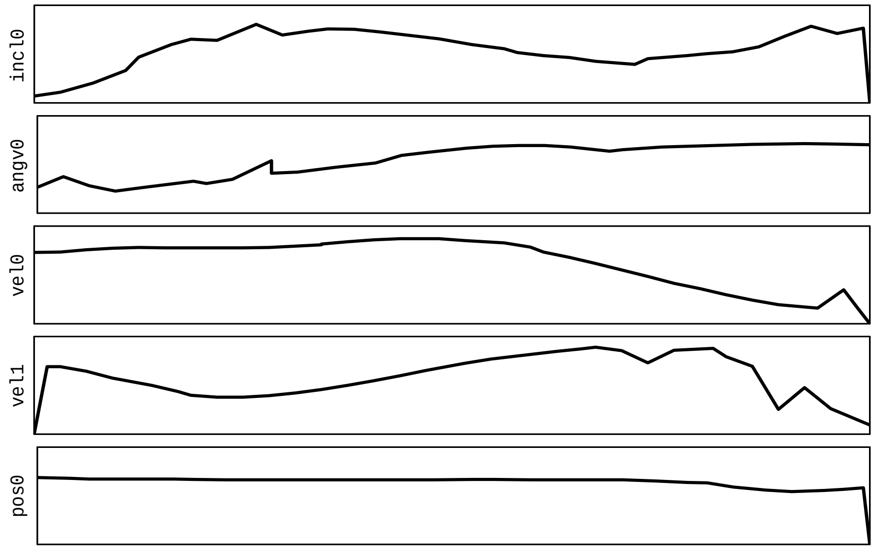
4.4 *Solutions*

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

In almost all 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 3 shows a sample controller evolved by the algorithm.

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

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.



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 3: 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

2. Push the ball toward the target interval with short, gentle tilts in the right direction, and immediately balance with opposing tilts. 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 cases observed, the last part of the strategy was not completely successful, in that every now and then the ball managed to escape from the target interval for a very short time before being pushed back into it.

5 Summary and Conclusions

This chapter has 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 specialized definitions of crossover and mutation were illustrated.

The proposed approach was tested on the ball-and-beam problem, and the results were qualitatively described. Since it might be objected that the ball-and-beam problem is not a real-world problem and it is too simple to show the effectiveness of the method proposed, work is underway to apply it to a difficult, highly non-linear power supply control problem.

The reader should note that the algorithm, in its present state, has purposely been designed to take advantage of neither problem-specific expert knowledge that may be available nor control engineering common sense. Improvements in these directions are always possible[a]. 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 hand-crafted approximations of the controllers sought.

In conclusion, evolutionary techniques have been shown capable of devising programs that make effective and efficient use of all degrees of freedom provided by a fuzzy microprocessor such as SGS-Thomson's W.A.R.P. The details of an evolutionary algorithm for the synthesis and optimization of W.A.R.P. program have been presented, and results of experiments have been qualitatively described. The implementation and use of a "fuzzy government" for monitoring and controlling such algorithm has been discussed. The adoption of a competition-based selection scheme that does not require the explicit definition of a fitness function has led to the definition of a fuzzy fitness for use in the fuzzy government.

Acknowledgements

The authors would like to thank all the people who made this research possible: The people at Co.Ri.M.Me. for help with W.A.R.P. documentation, Dr. Roger Jang for providing the dynamics of the ball-and-beam problem, Dr. Mike Lee for many stimulating discussions, Prof. Lotfi Zadeh for his valuable suggestions and Prof. Gianni Degli Antoni for his support.

[a]Note that two days of CPU time on a workstation are anyway less expensive than one hour of the time of a control engineer!

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)* (IEEE Computer Society Press, Los Alamitos, CA, 1994).
2. A. Bergmann, W. Burgard, and A. Hemker, "Adjusting parameters of genetic algorithms by fuzzy control rules," in K.-H. Becks and D. Perret-Gallix, editors, *New Computing Techniques in Physics Reserach III* (World Scientific Press, Singapore, 1994).
3. D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, (Addison-Wesley, Reading, MA, 1989).
4. C. Z. Janikow, "A genetic algorithm for learning fuzzy controllers," in *Proceedings of the ACM Symposium on Applied Computing* (ACM Press, New York, 1994).
5. 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* (Morgan Kaufmann, San Mateo, CA, 1991).
6. J. R. Koza, *Genetic Programming: on the programming of computers by means of natural selection* (The MIT Press, Cambridge, Massachussets, 1993).
7. M. Lee and H. Takagi, "Dynamic control of genetic algorithms using fuzzy logic techniques," in S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms* (Morgan Kaufmann, San Mateo, CA, 1993).
8. M. Lee and H. Takagi, "Embedding apriori knowledge into an integrated fuzzy system design method based on genetic algorithms," in *Proceedings of the 5th IFSA World Congress (IFSA'93)* (1993), Vol. II, pp. 1293–1296.
9. 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)* (1993), Vol. I, pp. 612–617.
10. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs* (Springer-Verlag, Berlin, 1992).
11. H.-P. Schwefel, *Numerical optimization of computer models* (Wiley, Chichester; New York, 1981).
12. SGS-Thomson Microelectronics, *W.A.R.P. Software Development Tool User's Guide*, version 1.0 (1994).

13. 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).
14. P. Thrift, "Fuzzy logic synthesis with genetic algorithms," in R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms* (Morgan Kaufmann, San Mateo, CA, 1991).