Evolutionary Design of Fuzzy Logic Controllers Using Strongly-Typed GP

Enrique Alba, Carlos Cotta & José Mª Troya
Dept. of Lenguajes y Ciencias de la Computación, Universidad de Málaga
Campus de Teatinos (2.2.A.6), 29071-Málaga (ESPAÑA)
{eat, ccottap, troya}@lcc.uma.es

Abstract

An evolutionary approach to the design of fuzzy logic controllers is presented in this paper. We propose the use of the genetic programming paradigm to evolve fuzzy rule-bases (internally represented as type-constrained syntactic trees). This model has been applied to the cart-centering problem, although it can be readily extended to other problems. The obtained results show that a good parameterization of the algorithm, and an appropriate evaluation function, can lead to near-optimal solutions.

Keywords: Genetic Programming, Type System, Fuzzy Logic Controller, Cart-Centering Problem

1 Introduction

Genetic Algorithms (GAs) [6] are heuristic optimization procedures based on the principles of natural evolution. They combine the concepts of adaptation and survival of the fittest to produce sub-optimal solutions for an arbitrary problem. This is achieved by means of the repeated application of mating and reproduction operators onto a population representing tentative solutions.

The Genetic Programming (GP) paradigm [10] [1] is a field of research focused on the automatic breeding of computer programs by means of genetic algorithms. The population of the GA is composed of syntactically correct trees, each one representing a computer program that solves a given task.

This task is presented to the GA as a set of examples that express the behavior the solution tree must exhibit. The tree is composed of a set of functions (internal nodes of the tree), and a set of terminal symbols (leaves) that together conform a sub-optimal solution to the problem.

In this work we try a new approach in which individuals (trees) are not computer programs in a traditional sense. Every tree will represent a set of fuzzy rules intended to solve a control problem (the cart-centering problem, described further in this paper). The underlying target programming language is neither C, PASCAL, nor LISP. We use a general fuzzy interpreter in order to execute the resulting fuzzy rules on our specific problem. We are interested in studying the new problems this field presents just as the power of the GP paradigm in solving a new kind of application.

There exist three fields of application of the evolutionary algorithm paradigms to the design of fuzzy logic controllers:

(1) Learning the semantics of the fuzzy sets, that is, evolutionary tuning processes [7] [8].

(2) Learning the rules with a fixed semantic [5] [14] [18].

(3) Learning the fuzzy sets and the rules at the same time (the full knowledge base) [2] [3].

In this contribution we propose a GP model for learning the rule base, that is, for learning the rules with a fixed semantic.

This paper is organized as follows: a brief review of fuzzy logic controllers and a description of the cart-centering problem is first outlined, followed by the GP model used for the experiments. Finally, we present a comparative performance analysis of our approach, and extract some important conclusions regarding the evaluation function and the reproductive policy.

2 Fuzzy Logic Controllers

Fuzzy Logic Controllers (FLCs) are rule-based systems that successfully incorporate the flexibility of human-decision making by means of the use of the fuzzy set theory [11]. Natural-language terms (e.g. pressure is *low*) are used in

fuzzy rules. Their ambiguity is modeled by membership functions, intended to represent human expert's conception of the linguistic terms. A membership function measures the degree of confidence by which a precise numeric value is described by a linguistic label.

Rules take the form **IF** [*conditions*] **THEN** [*actions*], where *conditions* and *actions* are linguistic terms describing the values of input and output variables (e.g. **IF** *pressure* **IS** *low* **THEN** *valve-opening* **IS** *small*). The fuzzy rule-base of the FLC is composed of a number of such fuzzy rules. This rule-base is used to produce precise output values according to the actual input values. This control process is divided into three stages (Figure 1):

a) *fuzzification*: calculate fuzzy input, i.e. evaluate input variables with respect to the corresponding linguistic terms in the condition side.
b) *fuzzy inference*: calculate fuzzy output, i.e. evaluate the activation strength of every rule and combine their action sides.
c) *defuzzification*: calculate actual output, i.e. convert fuzzy output into a precise numerical value.

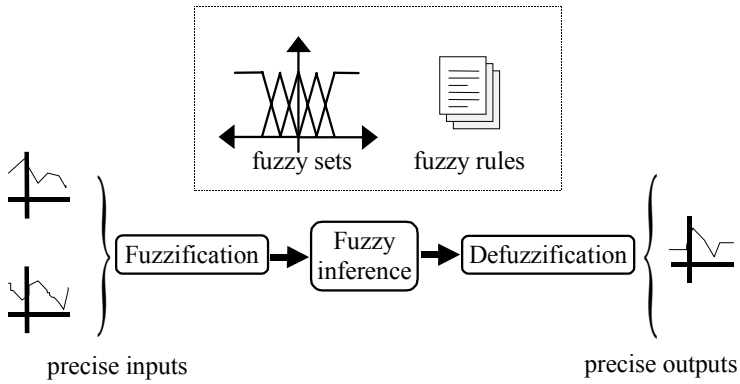


Figure 1. *The Fuzzy Control Process.*

The fuzzy interpreter used in this work performs fuzzification via triangular membership functions, uses the *min* intersection operator, the *maxmin* method for fuzzy inference, and the *centroid* procedure for defuzzification (for comparison purposes with [18]).

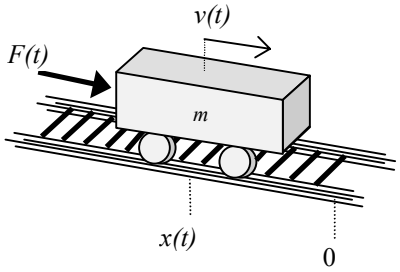
3 The Cart-Centering Problem

In this section a description of the cart-centering problem is outlined. Next, it will be shown how the principles of fuzzy control can be applied to this problem, outlining an ad-hoc solution.

3.1 Description of the Problem

The cart-centering problem definition involves a cart with mass *m* moving along an infinite one-dimensional frictionless track. The problem consists in centering the cart, in minimal time, by applying an external time-variant force (of maximal magnitude *F*) so as to accelerate the cart in either the positive or the negative sense of the track.

Given the values for the position *x(t)* and velocity *v(t)* of the cart at time *t*, this problem can be further formulated as calculating a force *F(t) = F[x(t), v(t)]* in order to place the cart in position $x(t_f)=0$ with velocity $v(t_f)=0$ in minimal t_f.



$$F(t) = \begin{cases} +F & \text{if } -x(t) > \dfrac{v(t)^2 \operatorname{sgn}\left[v(t)\right]}{2\dfrac{F}{m}} \\ -F & \text{otherwise} \end{cases}$$

Figure 2. *The Cart-Centering Problem and its optimal solution.*

Though simple, this is a typical optimal control problem that has a well-known solution (the *bang-bang* rule [10]). This system is simulated using Euler's method, i.e. taking time steps of τ sec. and applying Newton's Laws:

$$x(t+\tau) = x(t) + \tau \cdot v(t) \qquad v(t+\tau) = v(t) + \tau \frac{F(t)}{m}$$

As in [4] and [18], we consider τ=0.02 s and *m*=2.0 kg.

3.2 Designing a Fuzzy Logic Controller

As an optimal control problem, the cart-centering problem is well-suited for fuzzy control. The development of an FLC for this system involves the following three steps [8]:

a) Determine *condition* (input) variables, and specify the fuzzy sets describing them.
b) Determine *action* (output) variables, and specify the fuzzy sets describing them.
c) Design the rule-base.

It is clear that position *x(t)* and velocity *v(t)* of the cart are the input variables, and the force *F(t)* applied to the cart is the single output variable in this problem. The membership functions defining the fuzzy sets for these variables are shown in Figure 3: five partitions representing the linguistic terms negative large (NL), negative small (NS), zero (ZE), positive small (PS), and positive large (PL).

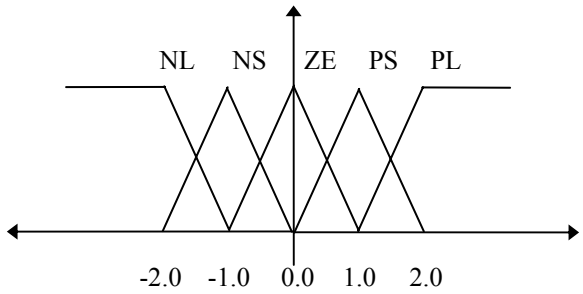


Figure 3. *Fuzzy sets for position, velocity and force.*

The third step (production of the rule-base) may be accomplished by an expert. The use of fuzzy linguistic terms allows an easier incorporation of human knowledge. For example, it seems intuitive that the applied force must almost always be directed towards the origin (since we want to place the cart in the 0 position) and, just when the cart is close to that point, the force must be reversed to stop the cart. The following fuzzy rules express these rules-of-thumb:

```
IF pos IS PL THEN for IS NL
IF pos IS PS THEN for IS NL
IF pos IS ZE AND vel IS PL THEN for IS NL
IF pos IS ZE AND vel IS PS THEN for IS NL
IF pos IS ZE AND vel IS NS THEN for IS PL
IF pos IS ZE AND vel IS NL THEN for IS PL
IF pos IS NS THEN for IS PL
IF pos IS NL THEN for IS PL
```

where *pos*, *vel* and *for* stand for position, velocity, and force, respectively.

The FLC described above is not the optimal one. Improvements in the rule-base or in the interpretation of the linguistic terms may be done by means of GAs (see [4], [8], [12], [18]). We propose the use of GP to find rule-bases that can be used as a starting point for further refinements by a human expert or even as a final solution.

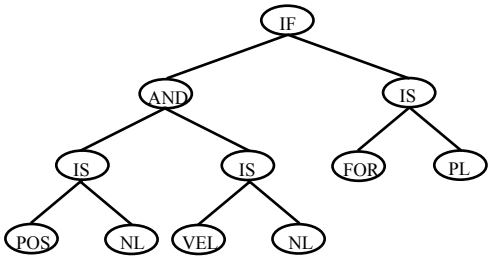
4 Genetic Approach for Learning Fuzzy Rules

In this section we propose a genetic programming model for designing the rule base, i.e. for learning the rules with a fixed semantic. First, the necessity of a type system is shown; second, some general details about the evaluation function are outlined.

4.1 Encoding Rule-Bases

Given that every individual contains a list of rules, an encoding for single rules and subsequently a mechanism to join them is needed. A single rule can be easily represented as a binary tree: the root being an `IF` node, and left and right branches representing the condition and the action sides, respectively. Likewise, both conditions and actions can be expressed as trees. On the one hand, a variable paired with a fuzzy set can be represented as a tree with an `IS` root-node: the variable name in the left branch and the fuzzy set in the right branch.

On the other hand, a conjunction of such terms can be expressed as a tree with an AND root-node and two branches representing nested conjunctions or pairs *variable* **is** *fuzzy_set*. Figure 4 shows an example.



IF *pos* **IS** NL **AND** *vel* **IS** NL **THEN** *for* **IS** PL

Figure 4. *A syntactic tree and the rule it represents.*

A list of rules might be represented as a list of trees. However, it is more appropriate to use a single tree representation because it allows rules to be tightly connected and therefore facilitates the propagation of good functional blocks. Two additional symbols are required. The EL symbol represents an empty list. The RLIST symbol combines rules in the same fashion in which the AND symbol joins terms, i.e. a list of rules is a tree with an RLIST root-node and two branches representing single rules or analogous lists of rules.

This is a very flexible and powerful representation of FLC rule-bases. It deals well with the fact that an FLC may have rules with very different structures among them as well as a variable number of rules. This aspect contrasts with other GA-based approaches in which the structure [12] [18] or the number of rules in the FLC [4] are arbitrarily prefixed.

4.2 Necessity of a Type-System

The flexibility of this tree representation makes crossover a very powerful tool since it allows interchange to occur at several levels: rules, terms or even variable names. However, the traditional GP crossover operator may produce nonsense rules; for example, consider the case in which a variable name is substituted by a whole rule. There are three possible solutions for that problem:

a) Define closed functions, so those ill-defined rules are reinterpreted in some predefined way [10]. For example, the `IS` function might consider anything different from a variable name in the left branch as a dummy variable with a fixed value.
b) Repair the tree, i.e. deleting and adding sub-trees whenever necessary.
c) Select crossover points so that the resulting trees be correct [15].

Since the first option will produce a high number of useless trees composed mostly of dummy rules, and the second one is computationally expensive and does not preserve causality due to the repairing mechanism [16], we have chosen the third option.

In our system, every symbol has a type attribute, determined by means of the partitions defined by the equivalency relationship "is interchangeable with". For example, since an `EL` node may be swapped with an `RLIST` node, they have the same type. Figure 5 shows all types.

TYPE	SYMBOLS
Type I	{ `IF, RLIST, EL` }
Type II	{ `AND, IS` }
Type III	{ ... variable names ... }
Type IV	{ ... fuzzy sets ... }

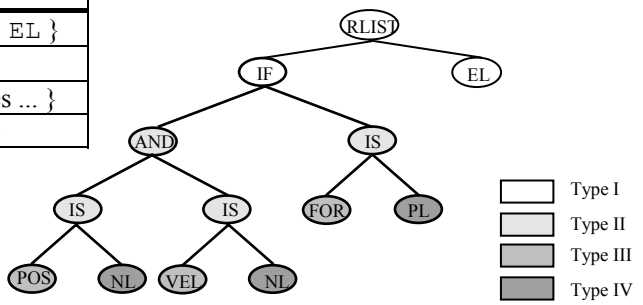


Figure 5. *Basic types and symbols belonging to each type.*

The crossover operator first selects a random node in one parent; then, a node of the same type is randomly selected in the other parent and the corresponding sub-trees are swapped, thus producing two valid offspring. However, considering just the types above leads to the creation of syntactically correct but semantically incorrect trees with output variables appearing in the condition side and/or input variables appearing in the action side (e.g. **`IF`** *`for`* **`IS`** `NL` **`THEN`** *`pos`* **`IS`** `ZE`). These situations must be avoided to make a good use of computational resources. To achieve this, two subtypes for both type II and III are defined.

Every subtype can be seen as the *variety* of the corresponding type that appears in either the condition side or the action side. For example, for type `VBLE` (type III), we define the subtypes `VBLEIN` and `VBLEOUT`, representing input and output variables respectively. See Figure 6 for a complete taxonomy of types and subtypes.

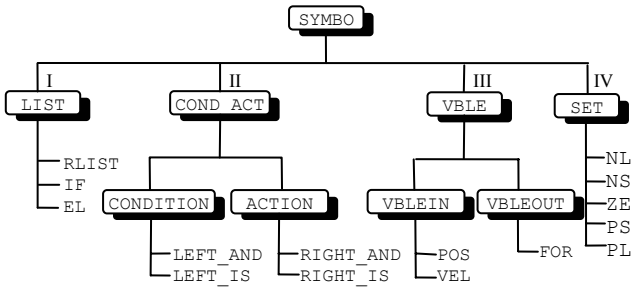


Figure 6. *Types and symbols used.*

4.3 Generating Rule-Bases

We have initially used an elitist generational GA to evolve trees. These trees are bounded to have a maximum depth of 6 levels in the initial generation and 15 in subsequent generations. The initial population is generated using a *half-ramping mechanism* (half of the population is composed of random complete trees with the given initial depth, and the other half is generated as random incomplete trees) [10]. We follow the BNF grammar shown in Figure 7 to produce correct trees. Notice that only the last two rules (describing the input and output variables of the problem) are problem-dependent. The rest of rules can be used in any other fuzzy controller.

Every tree is generated in a top-down fashion, i.e. choosing a root-node and then producing appropriate sub-trees in a similar way. The trees are implemented as variable length strings. To generate a node, a random type is chosen and subsequently a random symbol belonging to that type is assigned to that node. The selection of this random symbol depends on the level of the node (e.g. an `IF` node cannot appear in levels below 3 because its branches must hang two levels down -an `IS` sub-tree- at least). Figure 8 shows which symbols may be root of a sub-tree according to its parent and the maximal depth of the tree. The root node of the whole tree may be only one of the thick nodes.

```
<TREE>     ::= EL | <IF> | <RLIST>
<RLIST>    ::= <TREE> <TREE> RLIST
<IF>       ::= <COND> <ACT> IF
<COND>     ::= <L_IS> | <L_AND>
<L_IS>     ::= <VBLEIN> <SET> LEFT_IS
<L_AND>    ::= <COND> <COND> LEFT_AND
<ACT>      ::= <R_IS> | <R_AND>
<R_IS>     ::= <VBLEOUT> <SET> RIGHT_IS
<R_AND>    ::= <ACT> <ACT> RIGHT_AND
<SET>      ::= NL | NS | ZE | PS | PL
<VBLEIN>   ::= VEL | POS
<VBLEOUT>  ::= FOR
```

Figure 7. *BNF grammar of correct trees (trees are described and implemented as post-ordered strings).*

In every generation, the two best individuals are copied to the next generation. The rest of the population is obtained through crossover operations between selected individuals. This is done selecting 50% of the population according to scaled fitness and 50% randomly. This selection procedure has revealed itself as appropriate in other domains and represents a good tradeoff between exploitation and exploration.

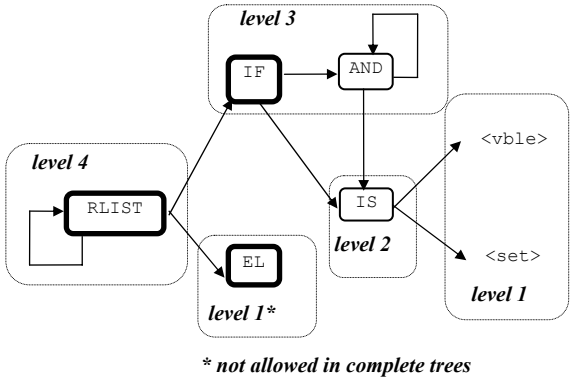


Figure 8. *Allowed argument types, and the minimal level at which they may appear in a complete tree.*

4.4 Evaluating Rule-Bases

Individuals are evaluated by averaging over 16 simulations. The initial conditions for each simulation are equally distributed points in the space-state.

This evaluation technique is preferred to selecting random starting points because it is less noisy and ensures a better coverage of all possible cases. However, there may be situations in which the computational complexity of a simulation makes it too expensive to perform such a complete sampling. In these cases performing random tests may be a good tradeoff.

The simulations are executed for a maximum number of 500 time steps. The cart is considered to be centered if it is within a small enough neighborhood of the target state. Notice that considering a particular distance function for defining the boundaries of the neighborhood notably influences the final results, as shown in the next section. If the cart could not be centered, the maximum number of steps is taken. The mean time of all simulations is subtracted from 500 to obtain a fitness value (in order to have a maximization problem).

5 Results

In this section we show the results our GP model has produced. For comparison purposes, we also include the results of the intuitive solution described in Section 3.2, and some results from related works.

In Section 5.1 we show the relationship between the optimal and the hand-made solutions to the problem. Section 5.2 proceeds in several steps. It begins by presenting a basic GP solution with a generational GA, then it discusses the influence of the evaluation function and the stopping criterion on the results. Finally, a steady-state GA [17]is used to analyze the influence of the evolution scheme.

5.1 Optimal and Intuitive Solutions

To measure the performance of a given FLC, the mean time to center the cart throughout 100 simulations of the problem, starting from random initial points in the domain (-2.5, 2.5) is used. Situations in which the initial state is a solution are excluded. As stated in Section 3.1, the parameters of the simulations were $\tau = 0.02$ sec, $m = 2.0$ kg. Force has a cut-off value of 2.5 N.

Simulations are executed for a maximum of 500 time steps. The cart is considered centered when $\mathbf{max}(|pos|, |vel|)<0.5$. Whilst the optimal solution averaged 129 time steps to center the cart, our *naive* solution took 249 time steps, timing-out 14 times. In Figure 9, their two control surfaces are shown.

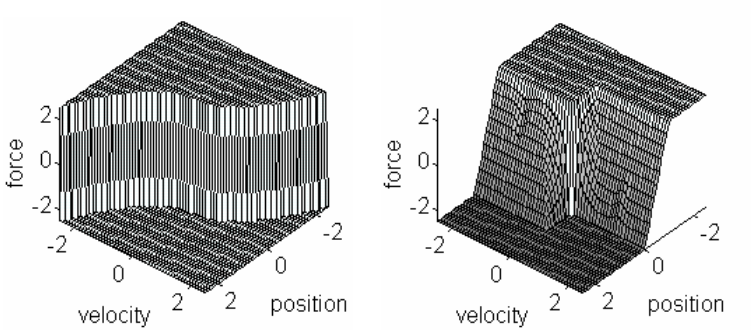


Figure 9. *Optimal* (left) *and naive* (right) *control surfaces.*

The 18 rule-FLC described in [18] is tested too with the mentioned parameters (Table 1). Its performance is better than our *naive* solution. The mean time to center the cart is 149 time steps.

5.2 The GP solution

Following [10], a population size of 500, for a maximum number of 51 generations (the initial one plus 50 additional generations) was chosen, and 10 independent runs were performed. The reproductive policy was generational and elitist. Figure 10 shows the best and the mean fitness in every generation, averaged for the 10 runs. See the parameters of the GA in Table 1.

TABLE 1. PARAMETERIZATION OF THE GENETIC ALGORITHM

Fitness cases	16 equally spaced points (x,v) taken from the interval [(-2.5, -2.5), (2.5, 2.5)].
Raw fitness	Mean time, over the 16 fitness cases, needed to center the cart. Timeout contribution is 500 time steps.
Selection	Elitist (2 best), 50% fitness proportional and 50% randomly.
Crossover	restricted for crossing only compatible types. Pc=1.0.
Mutation	none.
Replacement	Generational with elitism.
Parameters	M=500 individuals, G=51 generations.

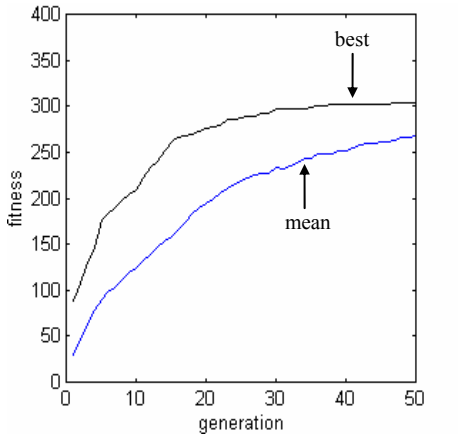


Figure 10. *Best and mean fitness averaged for 10 runs.*

The best generated individual obtained a fitness value of 321 (averaging 158 time steps to dock the cart), and corresponded to the following 9 rule-FLC:

```
1) IF pos IS PL THEN for IS NL
2) IF pos IS PS THEN for IS NL
3) IF pos IS NL THEN for IS PL
4) IF vel IS PL THEN for IS NL
5) IF vel IS NS THEN for IS PL
6) IF vel IS NL THEN for IS PL
7) IF pos IS ZE AND vel IS PS THEN for IS NL
8) IF pos IS NS AND vel IS ZE THEN for IS PL
9) IF pos IS PS AND pos IS PL AND vel IS ZE AND vel IS
   NS THEN for IS NL AND for IS NS AND for IS PL
```

The rule-base above was obtained after deleting duplicated rules. Rules #2, #3 and #6 appeared twice and rules #1 and #4 were repeated three times. This indicates they represent important functional blocks for the FLC. The best individual also contained two dummy rules whose condition side is equivalent to an empty fuzzy set, so they carry no influence on the output.

Notice the structure of rule #9. According to the grammar shown in Figure 7, this rule is syntactically correct. It is also semantically correct, since input variables only appear in the condition side and output variables are

located just in the action side. Fuzzy inference and defuzzification will give an interpretation to this rule. As stated, the GP system was intended to generate rule-bases for an FLC with fixed predefined fuzzy sets over inputs and outputs. However, *new fuzzy sets* can be expressed by means of the union and intersection of those predefined fuzzy sets, thus allowing their evolution.

In order to clarify the behavior of the naive, evolved and optimal solutions we show in Figure 11 their respective response surfaces. Notice the similitude of the second-row graphs between the optimal and evolved response.

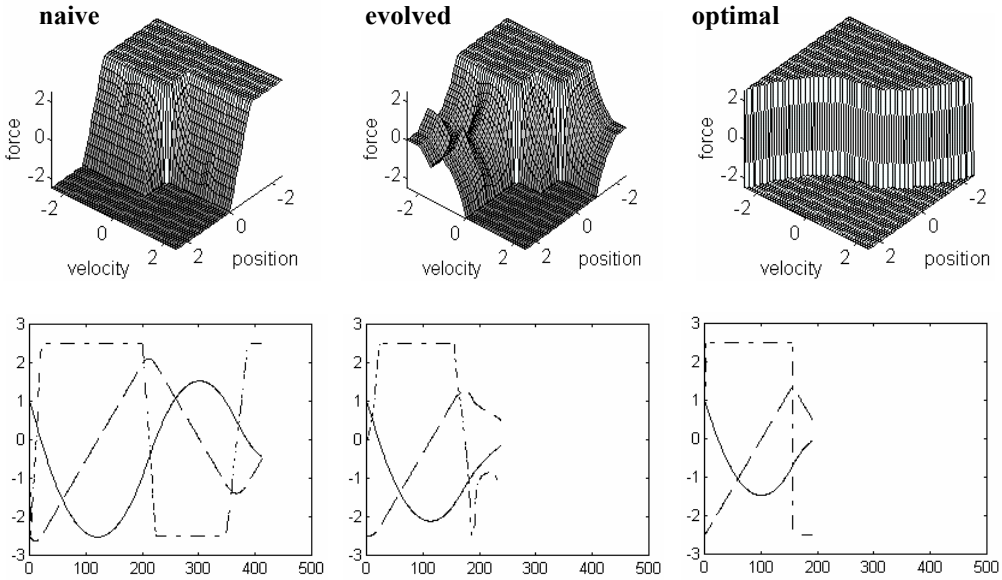


Figure 11. *Control surfaces and responses for pos* = 1.0 *m*, *vel* = -2.5 *m/s of the naive* (left), *evolved* (middle) *and optimal solution* (right). *The solid, dashed and dash-dotted lines represent position, velocity and force, respectively.*

However, looking at the first-row graphs it can be seen that it is a poor approximation of the optimal solution. Since the optimal output for this problem is always saturated, a wrapper converting positive values of *F(t)* to +*F*, and negative values to -*F* was added. The response surfaces of the non-saturated and the saturated versions are shown in Figure 12.

When a wrapper is used the average time to center the cart is reduced down to 131 time steps. However, this FLC did not evolve under that condition, and therefore the evolutionary process did not optimize it for that purpose.

Consequently, a new set of 10 runs was performed adding this wrapper to the evaluation function. Figure 13 left shows how the quality of the solutions is notably improved.

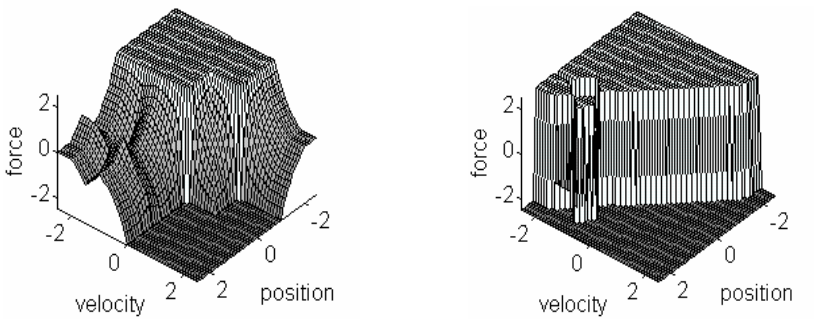


Figure 12. *Control surfaces of the evolved non-saturated FLC* (left) *and its saturated counterpart* (right).

The best evolved FLC with wrapper had 13 rules (see Figure 13 right). The average time it takes to center the cart is 120 time steps.

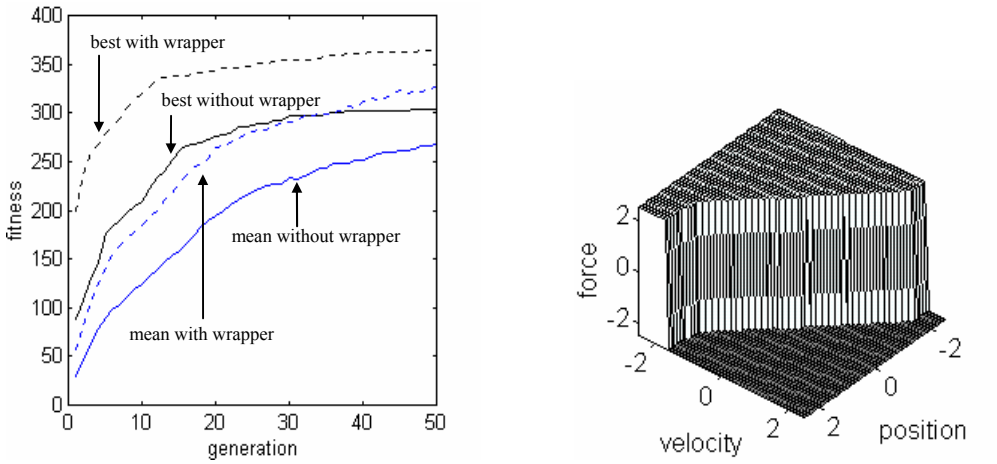


Figure 13. *Adding a wrapper: improvement in the fitness* (left)*, and control surface of the best evolved FLC* (right).

Though it may seem that this FLC performs better than the optimal solution (129 time steps), it is only a consequence of the used distance function. The **max** function considers as a goal state any point within a square centered in the origin. The GA learned that reaching a corner of the target zone is faster, in some situations, than approaching any other point of its perimeter. Figure 14 shows an example.

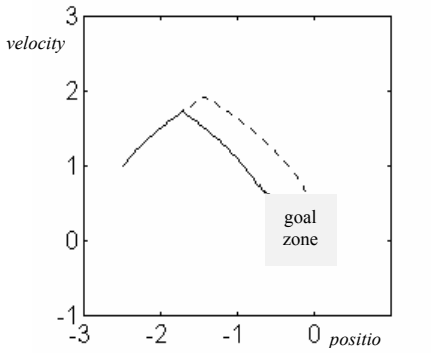


Figure 14. *Example for* $x(0)=-2.5$ *and* $v(0) = 1$ *of the optimal solution (dotted line) and the evolved FLC (solid line).*

If an Euclidean distance function is taken, the average time to dock the cart increases up to 133 time steps. To test the effects of this new stopping criterion, another set of 10 experiments were done. A new FLC with 21 rules was obtained. Its control surface is show in Figure 15.

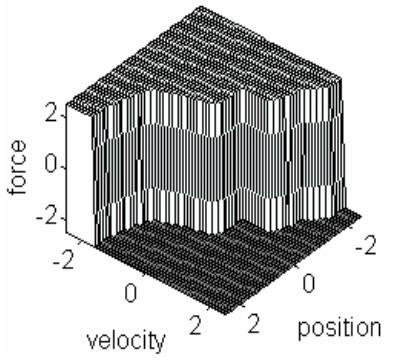


Figure 15. *Control surface of the best FLC evolved with a wrapper and Euclidean stopping criterion.*

This represents a very good approximation to the optimal solution. This FLC averages 127 time steps to dock the cart. The small *improvement* with respect to the optimal solution reflects the fact that the analytical solution is optimal to place the cart exactly in the origin, but not for moving it to an area around it.

Finally, since there exist some working hypotheses giving empirical advantages to a steady-state evolution [13] [17], a set of experiments were performed generating just 2 individuals in every iteration and inserting them into the population substituting the worst ones. Figure 16 shows a comparison of this model with the generational one (with wrapper and Euclidean distance). Notice how the steady-state GA has a higher convergence rate as expected, and therefore good solutions may be obtained earlier.

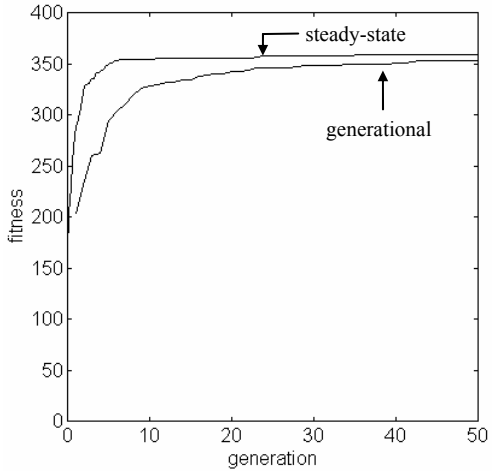


Figure 16. *Comparison of convergence rates in the generational and the steady-state models.*

6 Conclusions

A GP approach to FLC design has been presented. This class of genetic algorithms is more appropriate than other more traditional evolutionary models due to the use of high-level data structures; complex rule-bases may be easily represented using trees.

This encoding is much more flexible than the fixed-length binary encoding of simple genetic algorithms since there may be any number of rules in an FLC. Moreover, rules may have any structure as long as they are syntactically and semantically correct. Therefore, any good underlying (known or unknown) structure can evolve to improve the GP result.

This model has an additional advantage: its extensibility. Some GA-based systems work on representations in which the number or the structure of the rules are constrained. In such models, increasing the number of variables involved in the test problem implies a growth in chromosome length, and therefore large populations are required to keep diversity. Since most computational effort is devoted to simulate the system to obtain a fitness measure, a large population leads to very long runs of the GA. In our GP system, it would be necessary to increase the maximum depth of trees, thus allowing more complex rules and FLCs to evolve. However, since loss of diversity is not a main issue in GP [10], the population size could be kept constant (or just slightly increased). This also promotes using a fast evolution scheme like steady-state GAs.

The results are very promising. FLCs have been generated clearly outperforming an intuitive solution and other evolutionary approaches. Their quality is comparable with the optimal solution.

Future work could be directed to evolve more sophisticated FLCs including modifiers and credibility factors (as in fuzzy networks [4]). More difficult test problems (e.g. the cart-pole balancing) and novel GP techniques as ADF [9] for composed-set detection will be tried too.

References

[1] Banzhaf W., Nordin P., Keller R.E., Francone F.D., *Genetic Programming - An Introduction*. Morgan Kaufmann, 1998.

[2] Carse B., Fogarty T.C., Munro A., "Evolving Fuzzy Rule Based Controllers Using Genetic Algorithms". *Fuzzy Sets and Systems*, 80:273-294, 1996.

[3] Cordon O., Herrera F., “A Three-Stage Evolutionary Process for Learning Descriptive and Approximative Fuzzy Logic Controller Knowledge Bases

from Examples". *International Journal of Approximate Reasoning*, 17(4): 369-407, 1997.

[4] Feldman D.S., "Fuzzy Network Synthesis with Genetic Algorithms". In Forrest S. (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms,* Morgan Kaufmann, San Mateo CA, 312-317, 1993.

[5] González A., Pérez R., "Completeness and Consistency Conditions for Learning Fuzzy Rules". *Fuzzy Sets and Systems*, 96:37-51, 1998.

[6] Holland J.H., *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[7] Herrera F., Lozano M., Verdegay J.L., "Tuning Fuzzy Logic Controllers by Genetic Algorithms". *International Journal of Approximate Reasoning*, 12:299-315, 1995.

[8] Karr C.L., "Genetic Algorithms for Fuzzy Controllers". *AI Expert*, 26-33, 1991.

[9] Kinnear K.E., *Advances in Genetic Programming*, Chapters 5-6. MIT Press, Cambridge MA, 1994.

[10] Koza J.R., *Genetic Programming*. MIT Press, Cambridge MA, 1992.

[11] Lee C.C., "Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Parts I & II". *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):404-435, 1990.

[12] Lee M.A., Takagi H., "Dynamic Control of Genetic Algorithms using Fuzzy Logic Techniques". In Forrest S. (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo CA, 76-83, 1993.

[13] Levine D., "Genetic Algorithms: A Practitioner's View". *INFORMS Journal of Computing*, 9(3):256-259, 1997.

[14] Magdalena L., "Adapting the Gain of an FLC with Genetic Algorithms". *International Journal of Approximate Reasoning*, 16:335-358, 1997.

[15] Montana, D.J., *Strongly Typed Genetic Programming*. Bolt Beranek & Newman, Inc. Technical Report #7866, 1993.

[16] Rosca J.P., Ballard D.H., “Causality in Genetic Programming”. In Eshelman L.J. (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Francisco CA, 256-263, 1995.

[17] Syswerda G., “A Study of Reproduction in Generational and Steady-State Genetic Algorithms”. In Rawlins G.J.E. (ed.), *Foundations of GAs*, Morgan Kaufmann, 94-101, 1991.

[18] Thrift P., “Fuzzy Logic Synthesis with Genetic Algorithms”. In Belew R.K., Booker L.B. (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo CA, 509-513, 1991.