Learning a Fuzzy Controller by Genetic Algorithms

Cezary Z. Janikow
Department of Mathematics and Computer Science
University of Missouri St. Louis, USA
janikow@radom.umsl.edu

Catalin Buiu & Ion Dumitrache
Department of Intelligent Control and Bioengineering
"Politechnica" University of Bucharest, Romania
Spl. Independenti 313
buiu@roipb.bitnet & dumi@roipb.bitnet

KEYWORDS: Genetic algorithms, automatic process control, fuzzy rules.

Abstract

Fuzzy rules and inferences provide a powerful framework for controlling complex processes. Like symbolic AI systems, they offer a comprehensible representation, which facilitates transfer of knowledge between the system and human experts and users. Moreover, unlike most symbolic systems, the responses are more gradual. In this fuzzy representation, knowledge is distributed among three different levels: symbolic rules, numerical weights, and fuzzy linguistic definitions. When human expertise is used to build a fuzzy controller, it must often be tuned to fit particular objectives or simply be corrected when it is incomplete or otherwise incorrect. When such expertise does not exist or is difficult to acquire and model in the controller, the knowledge must be generated. In either case, it is desired to perform these tasks automatically. This can be accomplished by searching the space of all controllers, while sampling their performance to provide heuristic quality measures. This requires a robust search mechanism. Moreover, the search should be conducted simultaneously at all representation levels to avoid any assumptions about the knowledge. Genetic algorithms provide the necessary robustness. Their previous applications concentrated on manipulations of a single knowledge level. In this paper, we investigate their applications to manipulations of two different levels: the symbolic rules and the numerical weights. This is accomplished adapting a standard numerical genetic algorithm. A number of experiments is conducted and reported, which illustrate both tuning and learning capabilities of the system.

1 Preliminaries

Advancing microprocessor technologies have brought automation capabilities to new levels of applications. Process control is a very important industrial application area which can greatly benefit from such advancements. However, development and deployment of applications are often difficult because of complex dynamics of actual processes. Conventional control theory is based on mathematical models that describe the dynamic behavior of controlled systems. It is based on deterministic nature of systems, but its applicability is reduced by computational complexity, parametric and structural uncertainties, and presence of non-linearities.

These characteristics often make the controller design very complicated and unreliable. A potential solution is offered by shifting the attention from modelling of the process to extracting the control knowledge of human experts. This artificial intelligence approach brings the concept

of *Intelligent Control Systems* (ICSs), which utilize the fact that human operators normally do not handle the system control problem with a detailed mathematical model but rather with a qualitative, or symbolic, description of the controlled system. ICSs have two unique features: ability to make decisions and learning from data or experience. Decision making capabilities provide for the controllers to operate in real-time process-control environments, at both micro and macro levels of system operations. Learning capabilities make it possible for the controller to adapt its knowledge to given performance criteria, to reason about potential dynamics of the environment, to predict advantageous features, or even to acquire the needed knowledge. ICSs have the ability to react to a variety of environmental and durability related uncertainties.

Complex man-machine systems are characterized by a considerable amount of information. Conventional quantitative approaches are often inferior because of the mentioned process characteristics or simply for lack of comprehensibility. However, even operating on a qualitative structure proves to be deficient since the information is often imprecise and uncertain. *Fuzzy rules* provide a framework for application of linguistic constructs in a way that effectively utilizes the language uncertainties. A fuzzy controller consists of a set of fuzzy control rules with appropriate inference mechanisms [1].

The fuzzy controller is a more flexible alternative to a conventional PID controller. Technologically, it can be implemented with very little silicon surface. Special fuzzy logic chips are widely available. They can easily be interfaced to sensors and actuators. Moreover, explicit rules provide a more natural framework for expressing human expertise. However, due to non-linearities, the knowledge is difficult for tuning and adaptation. There are no standard procedures for doing that and normally time consuming human operation is needed. However, such procedures are necessary to provide the necessary learning and ease-of-operation capabilities that are vital for a successful application.

The research project presented here aims at investigating the use of *Genetic Algorithms* (GAs) for both extracting the necessary knowledge and tuning the existing knowledge of a fuzzy controller. Tuning an ICS-based fuzzy controller is a directed search through the representation space – in this case, spanned by weighted linguistic rules and fuzzy membership definitions. To be successful in complex environments, the search must be robust and informed. GAs provide the first property. Moreover, specialized GAs, utilizing rich task-knowledge, make the search heuristically directed.

Other approaches to learning fuzzy rules were similarly limited to very simple cases when the fuzzy search space could only be modified in a certain dimension (*e.g.*, [11]). On the other hand, a successful application would have to provide capabilities to modify fuzzy controllers simultaneously at the symbolic level of the rules and at the numerical levels of the weights and the fuzzy definitions. In this paper, we investigate genetic algorithm applications to tuning a fuzzy controller for a second order process by simultaneously manipulating the numerical weights and the symbolic rules structure. For the first level, we use a standard numerical genetic algorithm (Section 4.2). We then modify the algorithm to extend its operation to the other level (Section 4.3). Both approaches are illustrated with a number of experiments using a simulation process and various learning objectives. The final phase of the project, currently under investigation, involves modifications of the linguistic definitions as well.

2 Genetic Algorithms

Recently, we have seen an emergence of a new view of problem solving – interactions of active agents with the environment and the surrounding world. Some of these ideas are derived from nature, where organisms both cooperate and compete for resources of the environment in the quest for a better adaptation. Such observations led to the design of algorithms which simulate

these natural processes. One of the most successful such algorithms is so called genetic algorithm. GAs are adaptive search methods that simulate some of the natural processes: selection, information inheritance, random mutation, and population dynamics. The principles were first elucidated in [6], and since then the field has matured and enjoyed many successful applications. At first, GAs were mostly applied to numerical parameter optimizations due to an easy mapping from the problem to representation space [3]. Today, they find more and more general applications due to better understanding of the necessary properties the mapping must provide and new methodologies to process constraints.

2.1 Genetic Algorithms at a Glance

A GA operates as a simulation in which agents, organized in a population, compete for survival and cooperate to achieve a better adaptation. The agents are called *chromosomes* (higher-level organizations with multiple chromosomes are also possible). The chromosome structure (*genotype*) is made up of genes. The meaning of a particular chromosome (*phenotype*) is defined externally by the user. Traditional genetic algorithms operate on binary bits.

Genetic algorithms use two mechanisms to provide for the adaptive behavior: selective pressure and information inheritance. Selection, or competition, is a stochastic process with survival chances of an agent proportional to its adaptation level. The adaptation is measured by evaluating the phenotype in the problem environment. This selection imposes a pressure promoting survival of better individuals, which subsequently produce offspring. Cooperation is achieved by merging information from usually two agents, with the hope of producing more adapted individuals (better solutions) – this is accomplished by crossover. The merged information is inherited by the offspring. Additional mutation aims at introducing extra variability. Algorithms utilizing these mechanisms exhibit great robustness due to their ability to maintain an adaptive balance between efficiency and efficacy [3].

Figure 1 The GA algorithm.

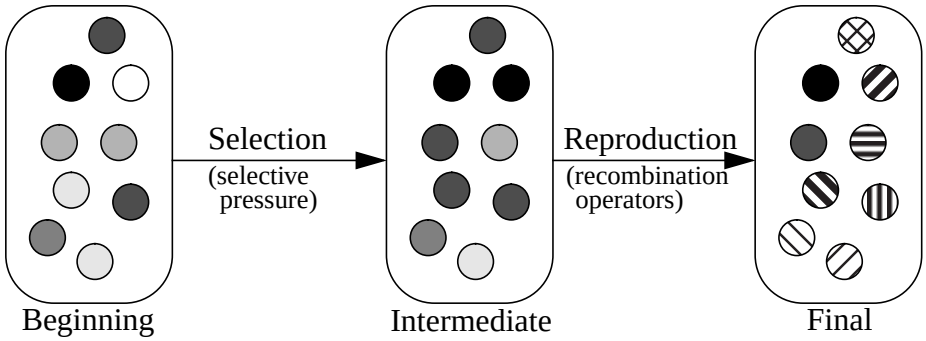
```
Generate initial population P(0)
Evaluate P(0)
While resources not exhausted and not done iterate
    Select P(t=t+1)
    Reproduce P(t)
    Evaluate P(t)
```

The simulation is achieved by iterating the basic steps of evaluation, selection, and reproduction, after some initial population is generated (see Figure 1). The initial population is usually generated randomly, but some knowledge of the sought solution may be used to an advantage. The iterations continue until some resources are exhausted. For example, the simulation may be set for a specific time limit or a fixed number of iterations. If some information about the sought solution is available, the simulation may continue until such criteria are met. Otherwise, the population dynamics may be observed and the simulation may stop if convergence to a solution is detected.

A single iteration is illustrated in Figure 2, where the bullets represent individual chromosomes and the shading levels indicate levels of adaptation – evaluation results. This evaluation is performed by a task-specific evaluation function. Each oval group represents the population instance at the iteration. Stochastic selection (with replacement) is applied to the beginning population instance, producing the intermediate state. Because of the selective pressure favoring survival of the better fitted individuals, the average fitness (darkness) of the chromosomes increases. However, no new individuals appear. Following the selection, reproduction operators are applied to members of the intermediate population. In this process, some chromosomes are

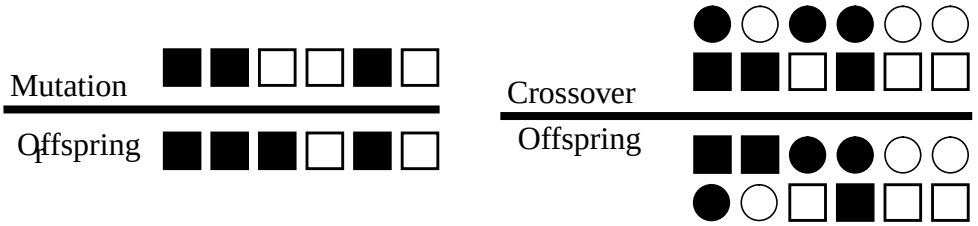
modified. Therefore, the final population instance will finally contain some new chromosomes. This iterative model is called the *generational* GA. Variations of this model are often used instead.

Figure 2 A single iteration of the generational GA model.



The two reproductive operators are visualized in Figure 3, which assumes binary coding for a chromosome (white and black genes). Mutation is performed here on the third bit of a the chromosome by flopping the *allele* (the particular instance of a gene). Crossover exchanges some genes between two chromosomes – here the exchange starts at the third bit.

Figure 3 Illustration of crossover and mutation.



Another mechanism is needed to apply the reproductive operators. A simple approach is to use a stochastic firing mechanism with some prior probabilities for mutation and crossover. A more sophisticated approach is to update these probabilities based on history, information contained in the population, or individual chromosomes. If the traditional crossover and mutation operators are used, the only relation to the process at hand is the evaluation function implementing the selection step. This is a great advantage, leading to domain-independent characteristics of the algorithm. This is also a great limitation, prohibiting the utilization of any available information about the problem [7].

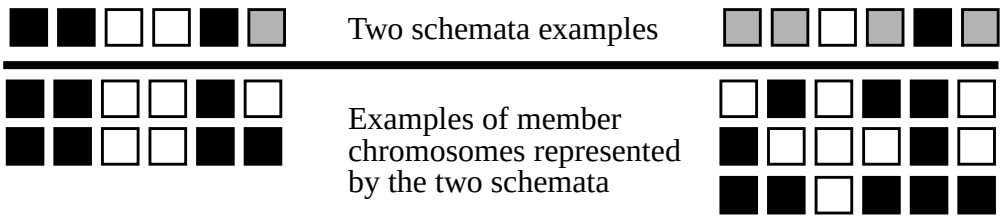
2.2 Theoretical and Intuitive Look at Genetic Algorithms

Genetic algorithms are not random searches. GAs explore regularities in the information the chromosomes represent. In a sense, the chromosomes are not really individuals but representatives of different species – two different chromosomes may really have similar adaptation levels if they represent similar species. However, the same two chromosomes may have different evaluations if the difference between them is significant. To explore such chromosome similarities, *schemata* are used. Schemata are similarity templates that contain fixed alleles for some genes but arbitrary alleles for others. For example, Figure 4 illustrates two different schemata – the shaded alleles represent *don't care* positions. The left schema represents species that can only have two different chromosome instances, as shown below it. The right schema is more general (actually, the left schema is a specialization of this one). A few of its representative chromo-

somes are shown below.

Unfortunately, the schema cannot be processed explicitly because they do not provide complete phenotype information needed for evaluations. Instead, a GA processes complete chromosomes. However, for practical problems, all possible chromosomes can not be processed. Therefore, the information about individual chromosomes is generalized to draw conclusions about above-average schemata.

Figure 4 Schemata illustrations.



The selective pressure causes the search to proceed by working with increasing representatives for the above-average schemata. The process continues by having more and more specific schemata represented in the population as the target templates. For example, if all the chromosomes on the right of Figure 4 evaluate high, a likely conclusion is that the third allele of the solution must be white and the fifth black. The schemata can also be seen as hyperplanes of the search space. A schema with no fixed positions is a hyperplane that spans the complete search space. A schema with only one fixed position is a hyperplane that halves the search space, *etc*. For example, the right schema of Figure 4, which has two fixed positions, represents exactly one-fourth of the possible number of chromosomes.

The iterative selection terminates when the represented schemata converge to a single most-specific schema – a fixed chromosome. However, no schema can be reached that was not represented in the initial population. To extend the search to other schemata, the reproductive operators are used. Therefore, reproduction causes exploration of new schemata as well as generation of new instances of the present schemata.

Unfortunately, both mutations and crossovers can disrupt the present schemata in addition to generating new ones. Given a proper balance between disruption and usefulness of the reproductive operators, the algorithm will continue exploration of better and better hyperplanes. Because of the trade-off and the limited resources normally available for the search, there is no guarantee that the globally optimal chromosome will be found.

The hyperplanes identified during the search as those above-average provide *building blocks* (the fixed positions) for the algorithm. Then, the same iterative search can be seen as a process in which very short building blocks (from very general schemata) are put together to form longer and longer blocks until a particular chromosome is generated. This hypothesis is called the *Building Block Hypothesis* [3].

Using building blocks, the reproductive crossover can be explained as a mechanism that assembles the building blocks identified by different chromosomes (and promoted by selection). Because the formation of such building blocks and the crossover's ability to propagate them depend on location of genes in the chromosome, the crossover will minimize disruption of schemata with short building blocks (short substructures). Mutation introduces much smaller disruptions, especially for the more-general schemata. These properties, along with the increasing representations for the above-average schemata, explain why genetic algorithms work and is called the *Schemata Theorem* [3].

Given this information, it is obvious now that the search properties will largely depend on reducing the disruptive properties and promoting formation of building blocks. These are largely

dependent on the mapping from the chromosome representation as a sequence of alleles to its problem-specific meaning. In particular, the location of a gene in the sequence proves to be very important. This led to theoretical analysis of different problems, with those not exhibiting "nice" building block properties classified as *GA-hard* [5]. For problems other than function optimization, the analysis is difficult.

Because GAs work by processing implicit schemata by means of explicit chromosomes, and the number of schemata having representatives in a population of some fixed size is exponential, genetic algorithms are said to exhibit *implicit parallelism* [3]. GAs also exhibit *explicit parallelism*, that is the processing can be parallelized. This property allows GA algorithms to utilize the fast-advancing parallel processing technologies [4].

2.3 GA Applications

GAs have been quite successfully applied to a number of problems, but by far they are famous as numerical optimization algorithms. The reason for this success is that numerical solutions can be easily represented as linear chromosomes, and both crossover and mutation are defined on such linear structures. Also, the quality assessment of such chromosomes is reduced to evaluating the original function. Moreover, most numerical problems exhibit nice building blocks – sequences of genes represent individual variables, and the sought solution is a vector of such variables.

Unfortunately, non-trivial GA applications require specialized representations. Moreover, the traditional mutation and crossover operators, even though suitable for any linearized representation, prove too blind for most practical problems. Instead, specialized operators are introduced for each specific representation, which implement some knowledge about the problem at hand [2].

3 Fuzzy Control

In classical set theory, an element either belongs to a certain set or does not. In this case, boolean processing can be employed resulting in crisp systems. However, in the real world, such scenario is often unrealistic because of insufficient language, imprecise measurements, *etc.* It is widely believed that some numerical component must be employed to accommodate such problems. However, purely quantitative approaches lack other characteristics desired of a successful approach. In particular, the most often cited disadvantage is lack of human-level comprehensibility and, what follows, difficulties with transfer and other usage of knowledge. Therefore, a successful learning system must combine quantitative and qualitative representation and reasoning. *Fuzzy sets* provide an interesting alternative.

In fuzzy sets, a fuzzy subset A of the U universe of discourse is described by a membership function $\mu_A(x)$: $x \in U \rightarrow [0,1]$, which represents the degree of x's belonging to set A. A fuzzy linguistic variable is an attribute whose domain contains linguistic terms – labels for fuzzy subsets. Fuzzy rules provide comprehensible knowledge representation that use fuzzy sets to deal with imprecision and can easily accommodate other linguistic terms. Along with some inference methods, they provide the interface between the rule-based knowledge and non-crisp decision-making [12].

In fuzzy rules for control, symbolic rules have conjunctive conditions and single conclusions. Different rules with the same conclusion represent implicit disjunctions. In addition, the rules may be weighted. The knowledge of a fuzzy controller utilizing this representation is distributed among three representation levels: symbolic level of rules, numeric level of rule weights, and numeric level of fuzzy linguistic definitions. Among these levels, there are complex interactions. For example, a rule with weight w=0 is effectively equivalent to no rule at all

if the inference mechanism takes the weights into account. To deal with this complexity, some obvious preference criteria can be introduced. The symbolic level should be as simple as possible since its complexity reflects on knowledge comprehensibility. The previous also implies that the numerical weight structure should be simple since such simplifies the symbolic structure. Finally, the numeric level of fuzzy definitions should be simple for both computational and comprehensibility reasons.

4 GA Applications to Learning Fuzzy Controllers

For experimentation, we selected a second order simulation process simple enough so that knowledge represented in a controller could be expressed graphically. This process was controlled by the fuzzy controller. We developed a genetic algorithm to first tune the controller to specific objectives by modifying the numerical weights, and then to learn the knowledge structure by also modifying and selecting the rules. The algorithm used a problem-specific representation – set of linguistic rules – and operators designed to manipulate at the representation.

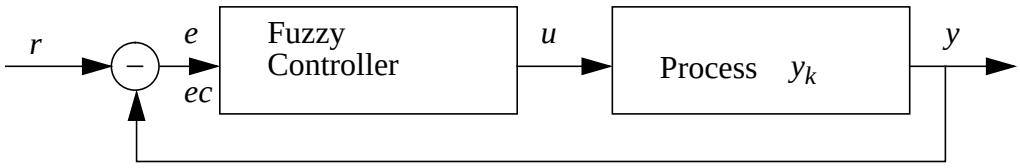
4.1 Simulation Problem and Knowledge Representation

We used the following second order simulation for the plant

$$y_k = 1.53y_{k-1} - 0.566y_{k-2} + 0.0255u_{k-1} + 0.015u_{k-2}$$

In the equation, *y* is the process output and *u* is process control. The momentum achieved by including past history ensures a modest complexity of the problem. The objective was to keep the output at a specific reference value which may unpredictably change at any time. The process can be manipulated by means of the control only. Since it takes time to match the output with the reference, some temporal error is inevitable. Therefore, a number of additional criteria must be considered, such as reduce overall error in a time frame, reduce the error fast, reduce overshoot, reduce oscillations, *etc.*

Figure 5 The simulation environment.



In our case, the single control value is computed by the fuzzy controller based on its knowledge of the process; see Figure 5, where *r* is the desired reference output, *y* is the current output, *u* is the action of the controller in response to the current error *e* and error change *ec*. The sought knowledge is a response surface: *u=f(e,ec)*. Because we did not assume any knowledge of the process itself, we could not assume any parametric form for it. Instead, this knowledge was represented in the fuzzy rules, weights, definitions, and response computation method. The fuzzy rules were of the form

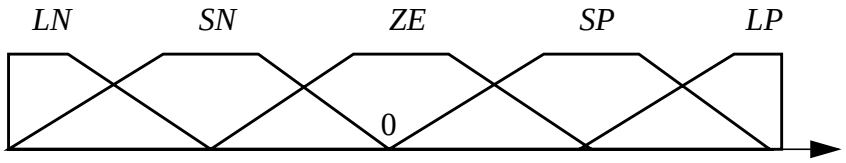
if *error* is LV_e and *error change* is LV_{ec} then *change in control* is LV_a : weight *w,*

which means that if the error has the linguistic value LV_e and error change has the linguistic value LV_{ec} then the control should be modified by the specified linguistic value LV_a – however, the relative importance of the rule is additionally modified by the weight *w.* For the linguistic values, we used the same fuzzy sets for all the error, error change, and control specified in a predetermined range and scaled to fit a particular problem– they are presented in Figure 6.

The knowledge is represented at three different levels: symbolic rules, weights, and fuzzy

definitions. Therefore, modification of a controller's knowledge should modify all three levels. Moreover, because of the complex relations among these levels, the modifications should apply simultaneously. In this paper, we investigate GA applications to modifying the first two. This combination is actually very challenging since these two levels are completely incompatible: one is symbolic, the other numerical. However, the problem may be greatly simplified if we utilize the interdependencies between these two levels and simply extend the numerical operators to modify the symbolic structure. The algorithm based on such ideas can easily be extended to the third level, that of the fuzzy definitions. Modifications of these can be seen as numerical modifications with some strong constraints (for example, for the trapezoidal definitions, the left corner of a trapezoid cannot be moved past the right one) and weak constraints (for example, keep the overlap "reasonable") – we plan to adapt the constrained operators introduced in the GENOCOP system [9].

Figure 6 The linguistic values and their fuzzy sets.



For the inference, we used the center of gravity method, which, for the trapezoidal linguistic values, is defined as: $\delta = \sum w_i \theta_i \alpha_i \zeta_i / \sum w_i \theta_i \alpha_i$, where w_i and θ_i represent the weight and activation degree of rule i, α_i and ζ_i are the area and centroid of the linguistic consequence, and δ is the resulting decision. The numerical decision can be converted to a single symbolic decision, or to a numerical vector, if desired. Other fuzzy reasoning and inference methods [10] are planned for the future.

4.2 Weights Tuning

The easiest knowledge modification level is that of the numerical weights. Since the weights represent a set of parameters, any parameter optimization genetic algorithm should be applicable. For the experiment, we started with a set of nine rules presented by a human expert and implemented in a fuzzy controller, which used the definitions of Figure 6 and *max* with center of gravity for the inference. The rules are shown in Table 1 – all were initialized with weight 1. It can be noted that this set does not include *LN* and *LP* linguistic values. This, however, does not reduce the applicability since the other three domain values cover the whole numerical domain space. In this experiment, our objective was to tune this knowledge by only modifying the weights. In other words, we assumed that the symbolic knowledge (the rules) and the fuzzy definitions were correct or at least not subject to modifications. Since the objective was to modify numerical values in order to optimize an objective function, we used a genetic algorithm designed for numerical optimization. The algorithm used a fixed population of solutions (vectors of weights), with values represented as constrained (in domain) floating point values. It also used the generational model in which first fitness-based selection generates a new population and then genetic operators stochastically modify some population members. The operators used included random mutation (in the domain), non-uniform mutation – designed so that the probability distribution for the mutated floating point value denses near the value to be mutated as the population ages, single crossover – with crossover points between individual weights, and arithmetical crossover – averaging selected weights (see [8]). For the evaluation function, we used different objective functions reflecting various criteria:

1. $J_1 = \sum_{t=1}^{T} e_t^2$ – designed to reduce overall error between desired and actual outputs.
2. $J_2 = \sum (e_t^2 + \alpha \times ec_t^2)$ – designed to also reduce the overshoot.
3. $J_3 = \sum (e_t^2 + \beta \times u_t^2)$ – designed to also reduce the control energy by reducing needed control changes.
4. $J_4 = \sum (t^2 \times e_t^2)$ – designed to reduce both error and oscillations, by additionally penalizing persistent errors in the time frame.

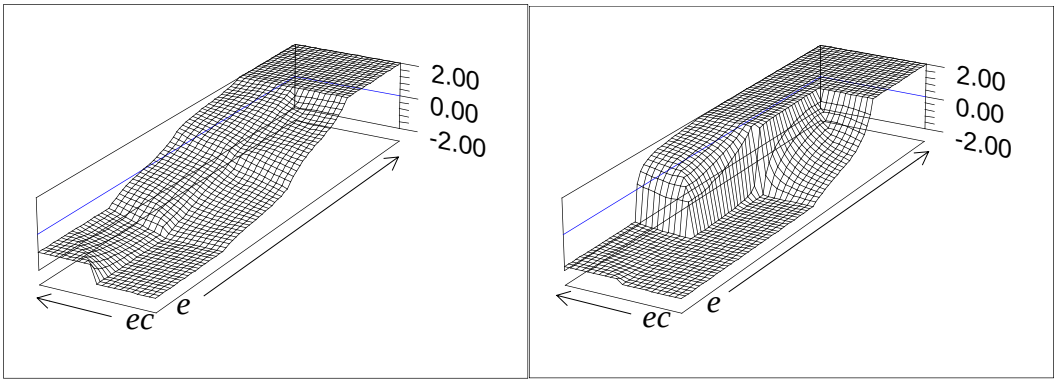
Each chromosome (a complete fuzzy-rule controller), in a given GA iteration, was evaluated by running an experiment (see Figure 5) for 200 time frames for the same initial state: y=0 and r=5. Because the state was being constantly changing after each control application in the time frame, rules to handle different states were needed. Nevertheless, this evaluation tended to learn better for positive errors r-y. For the future, we plan to extend this scenario to selecting a random situation for each population and accumulating the simulation results. THis non-stationary approach would also require departing from simple the generational model.

Table 1 Human expert generated rules – entries are linguistic changes in control *u*.

		Change in error *ec*		
		SN	*ZE*	*SP*
Error *e*	*SN*	*SN*	*SN*	*ZE*
	ZE	*SN*	*ZE*	*SP*
	SP	*SP*	*SP*	*SP*

First, we attempted to optimize the expert-provided fuzzy controller so that the accumulative quadratic error is minimized (J_1). Despite only few hundred GA iterations (200 in all of the reported experiments), a significant improvement was observed: J_1(y=0,r=1) index was reduced from 254 to 151. The response surfaces for the expert and J_1-optimized nine-rule controllers are illustrated in Figure 7 (all responses are shown for *e*'s universe and *ec*'s central quarter only since error changes rarely fell outside this region). It can be observed that the improvement was due to more rapid changes in response for low errors and to stronger differentiating between negative and positive error changes.

Figure 7 Response surfaces for expert and J_1-optimized nine-rule controllers.



How do the modifications actually translate into controlling the output of the process? In Figure 8 we trace their controlled outputs for the initial situation y=0, r=1. It can be observed that the output of the optimized controller reaches the desired reference faster and that the over-

shoot is actually smaller. However, a potential (depending on desired criteria) disadvantage of the controller is the output oscillation. Such oscillations would inevitably require continuous control adjustments even after the output is brought to the desired level. This may be prohibitive when, for example, control adjustments require high energy or cause undesired disruptions. The first trace of Figure 9 illustrates this problem. A number of potential solutions can be implemented. A simple one is to modify the fitness value to include changes in control *u* as well. This idea was implemented in the index J_3– so optimized controller requires much reduced control energy as illustrated in the second part of Figure 9.

Figure 8 Output *y* traces for expert and J_1-optimized nine-rule controllers.

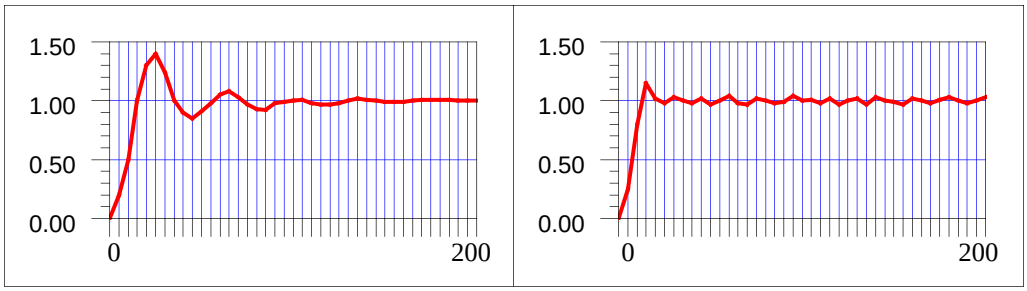
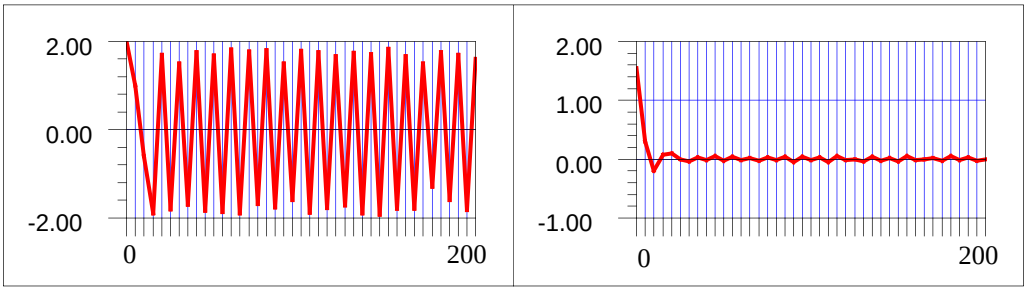
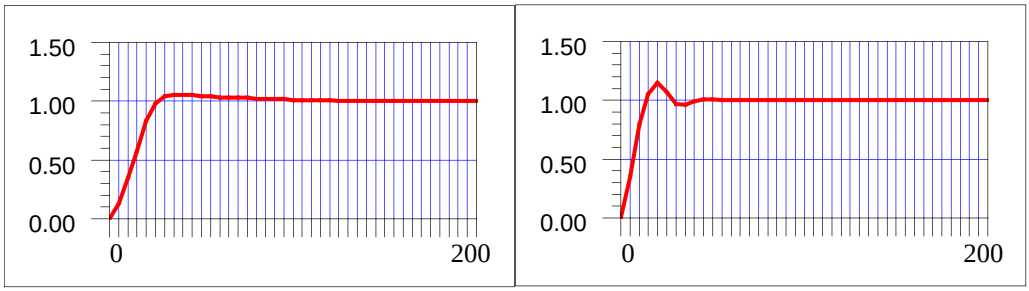


Figure 9 Change in control *u* traces for J_1 and J_3-optimized nine-rule controllers.



The same problem may be approached differently. For example, observing that the rule "if *e=ZE* and *ec=ZE* then *u=ZE*" is very important for stability at saturation, we might exclude its weight from modifications. Yet another approach might be to include a measure of oscillations directly in the fitness function – J_4 implemented that. Both approaches proved to provide the sought characteristics, as illustrated by lack of oscillations in Figure 10.

Figure 10 Output *y* traces for J_1-optimized with $w_{ZE,ZE,ZE}$=const and J_4-optimized controllers.



4.3 Weights Tuning and Rules Selection

In the previous section, we used a simple genetic algorithm to tune the knowledge of a controller, designed by a human expert, by manipulating only the numerical weights. The initial knowledge, even though not optimal, was complete (because of overlapping, the nine rules covered all possible error and change in error measures – with the applied scaling) and nonredundant (each

condition combination had only one possible action in the rule set). However, in general, one may not expect such friendly conditions since the expert's knowledge might be partial, erroneous, or even simply not available. Therefore, in this section we study genetic algorithm applications to such more general cases.

In the first experiment here, we assumed that the expert's knowledge existed but was noisy. This case happened to be nicely handled by the algorithm of Section 4.2. To obtain the initial controller, we took the same nine rules and corrupted them by adding ten additional random rules (still excluding *LN* and *LP* error and error change values) with default weighs w=1. The additional rules were such that some of the same conditions would have different actions. Then, we optimized the controller using 200 iterations and performance index J_4. Again, we obtained a sizable reduction in the value of the index, which is illustrated in output traces for y=0, r=1 in Figure 11. The corresponding response surfaces are plotted in Figure 12.

Figure 11 Output traces for initial and J_4-optimized 19-rule controllers.

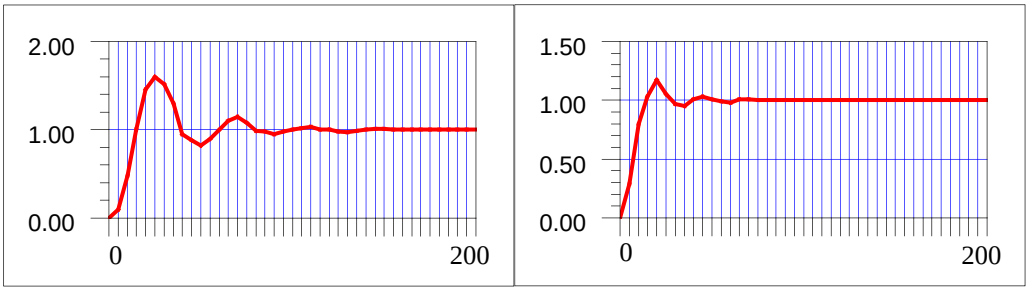
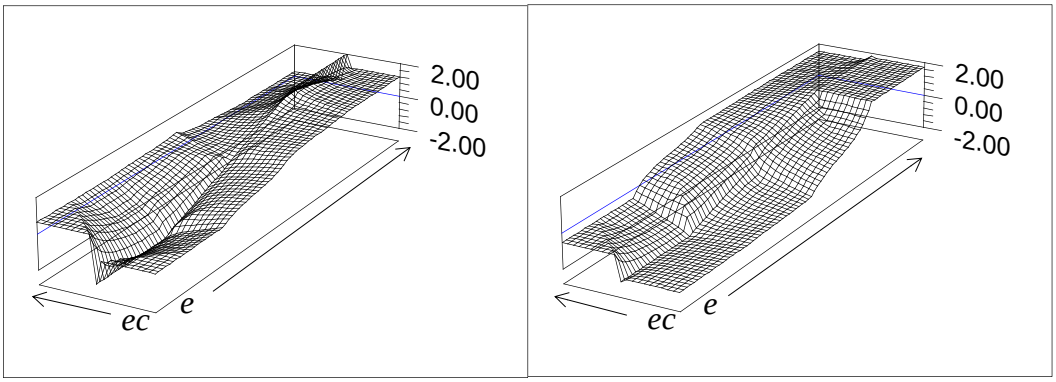


Figure 12 Response surfaces for initial and J_4-optimized 19-rule controllers.

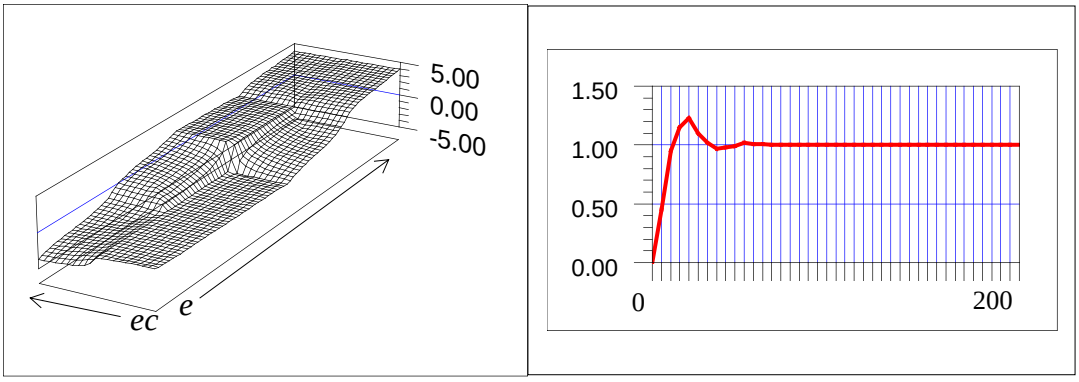


A much more interesting and challenging case is when the expert's knowledge is not available at all. Then, the controller must learn both the weights and the rule structure. To avoid any assumptions on the knowledge structure, we must provide for the algorithm to operate at the symbolic rule level to extract appropriate rules and at the weights level to optimize the choice of rule structure. One may accomplish the optimization objectives by still utilizing the same basic algorithm of Section 4.2. However, this approach would inevitably generate incomprehensive knowledge, especially given the limited number of iterations. Therefore, we decided to extend the algorithm to directly manipulate the rule structure as well. Because of the complex interdependencies between the two representation levels, one must operate on both levels simultaneously. There exists a number of ways for symbolic rule manipulation. However, to avoid the complexity of combining two different classes of operators, we decided to provide new numerical operators designed to simulate the needed operations at the symbolic level. This was possible due to the relatively simple rule structure (no internal disjunctions) and the fact that w_i=0 evaluates equivalently with the rule i absent from the structure.

To accomplish that, we defined a new mutation operator *to0*, which unconditionally modified a single weight to zero. This, given our inference, is equivalent to removing the corresponding rule from the controller. We also defined two other mutations *try0* and *try1*, which modified a weight to zero and one, respectively, but their application was conditioned by the weight being modified. For example, weight w_i=0.05 would have a high chance of undergoing *try0* – which would bring the weight to zero – and a very small chance of undergoing *try* – which would bring the weight to one. These operators were introduced to further reduce the rule structure in order to make it more comprehensive. Note that availability of the previously mentioned random and non-uniform mutations would allow a removed weight (by *try0* or *to0*) to be re-introduced into the set by modifying again the same weight. The same applies to *try1* (the purpose of *try1* was to simplify the structure by having as many fully weighted rules as possible).

In the experiment, we started with all possible rules (125). The operator *to0* had the highest application probability. As the measure of fitness we used J_4. Moreover, to facilitate simplification of the symbolic level, we modified the measure so that fewer rules would be preferred over more rules: $J_4^n = J_4 \times (1 + n/125)$, where n is the number of rules in the controller. These simple new operators and the modified performance index proved to be sufficient to optimize both the controller's performance (see Figure 13) and dramatically increase knowledge comprehensibility – the final number of rules was dropped to 12, with half of them having weight w=1.

Figure 13 Surface response and output trace for 125 rules trained with J_4^n.



5 Conclusions and Future Research

Fuzzy controller provides a powerful representation framework for controlling complex processes when implemented in fuzzy controllers. A controller must normally be tuned to fit dynamic objectives, their knowledge must often be corrected or even automatically generated when not available. In this paper we investigated genetic algorithm applications for all these tasks. This approach differs from previous, which concentrated on adapting the knowledge in certain dimensions only (*e.g.*, [13][14]).

Knowledge of a fuzzy controller has three distinct but strongly interdependent representation levels. Simple tuning can be accomplished by modifications of just one of these levels. However, in general, two or three levels must be operated on while searching for an optimal controller. In this paper, we first experimented with tuning a simple controller, generated using human expertise, to fit different objectives by only manipulating the weights level. Presented results indicate that this could be easily accomplished by an application of a simple numerical genetic algorithm used before for other parameter optimization tasks. Then, we extended the algorithm to simultaneously search the level of symbolic rule structures as well. This was accomplished by defining new numerical operators with specific symbolic interpretations. Again, presented results indicate that such modifications could lead to optimizing the controller

both in terms of simplifying the symbolic structure (and thus increasing its comprehensibility) and optimizing the performance. In the near future, we plan to further extend the algorithm to allow simultaneous manipulations of the third level – that of fuzzy definitions. This can be accomplished by utilizing some existing operators designed for constrained parameter optimization.

6 References

[1] Buckley, J.J. & Ying, H., *"**Expert Fuzzy Controller**"*, *Fuzzy Sets and Syst*ems, 44, 1991, pp. 373-390.

[2] Davis, L. (ed.), ***Handbook of Genetic Algorithms***, Van Nostrand Reinhold, 1991.

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

[4] Gordon, J.V.S., Whitely, D. & Bohm, A.P., *"**Dataflow Parallelism in Genetic Algorithms**"*,. In R. Manner & B. Manderick (eds.) *Parallel Problem Solving from Nature, 2*. North-Holland, 1992, pp. 533-542.

[5] Grefenstette, J.J., *"**Genetic Algorithms and Their Applications**"*. In A. Kent & J. Williams (eds.) *The Encyclopedia of Computer Science and Technology.* Marcel Dekker, 1990.

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

[7] Janikow, C.Z., *"**A Knowledge–Intensive Genetic Algorithm for Inductive Learning**"*. *Machine Learning* , 13 (2/3), pp. 189-228.

[8] Janikow, C.Z. & Michalewicz, *Z., "**An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms**"*. In Richard K. Belew & Lashon B. Booker (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 31–36.

[9] Michalewicz, Z. & Janikow, *C.Z., "**Handling Constraints in Genetic Algorithms**"*, In Richard K. Belew & Lashon B. Booker (eds.) *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 151–157.

[10] Mizumoto, M., *"**Fuzzy Controls Under Various Fuzzy Reasoning Methods**"*, *Information Science*, 45, 1988, pp. 129-151.

[11] Shao, S., *"**Fuzzy self-organizing controller and its application for dynamic processes**"*, *Fuzzy Sets and Systems*, 26, 1988, pp. 151-164.

[12] Sugiyama, K., *"**Rule-based Self-organizing Controller**"*, In M.M. Gupta & T. Yamakawa, eds.) *Fuzzy Computing*. North-Holland, 1988, pp 355-364.

[13] Urbancic, T., Juricic, D., Filipic, B. & Bratko, I., *"**Automated Synthesis of Control for Nonlinear Dynamic Systems**"*, In *Proceedings of International Symposium on Artificial Intelligence in Real-Time Control*, 1992.

[14] Valenzuela-Rendon, M., *"**The Fuzzy Classifier System: A Classifier System for Continuosly Varying Variables**"*. In Richard K. Belew & Lashon B. Booker (eds.) *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 346-353.