Genetic Algorithms in Process Identification and Control

Ion Dumitrache
Department of Intelligent Control and Bioengineering
"Politehnica" University of Bucharest
313 Splaiul Independentei
77206 Bucharest
ROMANIA

Catalin Buiu
Department of Intelligent Control and Bioengineering
"Politehnica" University of Bucharest
313 Splaiul Independentei
77206 Bucharest
ROMANIA

Abstract: Genetic algorithms are stochastic adaptive algorithms whose search method is based on simulation of natural genetic inheritance and Darwinean struggle for life. In the last years, there has been much interest in studying the theoretical foundations of genetic algorithms, and especially novel applications. The purpose of this paper is twofold. First, a survey of genetic algorithms is made, discussing what they actually are, and how they work. Secondly, some applications in process identification and control are presented, together with a practical application in fuzzy control.
Keywords: genetic algorithms, natural evolution, process identification, process control.

1. Introduction

In the 50s, von Neumann produced the theory of self-reproducing automata [1], where genetic algorithms (GAs) originated from. Holland continued this idea in [2]. In [3] the author discussed on the ability of a simple bit-string representation to encode complicate structures and on the transformations improving them. His work mainly demonstrated that, with a proper control structure, fast improvements of bit-strings could occur under certain transformations. GAs ([3], [4], [5], [6], [7], [8], [9], [10]) implement such ideas; they are a class of probabilistic algorithms that start with a population of randomly generated feasible solutions. The solutions "evolve" towards better ones by applying genetic operators modelled on the genetic processes occurring in nature.

GAs have been quite successfully applied to optimization problems like wire routing, scheduling, game playing, cognitive modelling, transportation problem, travelling salesman problem, adaptive and optimal control, fuzzy control problems, pattern recognition, neural networks, machine learning, etc.
This paper has two purposes. First, it provides a survey of GAs. We discuss what they are, and how they work. Secondly, there are reported some applications of GAs in process identification and control.

The paper is organized as follows. Section 2 presents the theoretical foundations of GAs. Section 3 describes some applications, and also the major problems encountered on implementing GAs, and some possible solutions to these problems. Section 4 gives an example of a GA applied to a fuzzy control problem. Section 5 draws conclusions and points to directions for future work.

2. Genetic Algorithms

Genetic algorithms (GAs) belong to a class of adaptive algorithms whose search methods are based on simulation of natural genetics and survival pressure. They pronouncedly differ from random algorithms in that they combine elements of direct and stochastic searches. If associated with difficult optimization problems they get superior to hill-climbing methods by simultaneously providing the best solutions and exploring the search space. Such a property renders them robuster than the existing directed search methods. The genetics based search methods characterize by as an important property as domain-independence.

In general, a GA performs a multi-directional search by maintaining a population of potential solutions, and encourages information generation

and exchange between these directions. This population enters a simulated evolution : on each generation, relatively "good" solutions proliferate, while relatively "bad" solutions expiate. In order to evaluate a solution quality, a special function, playing the role of the environment , is used. The structure of a simple GA is shown in Figure 1.

Running a genetic algorithm is to iteratively simulate by a global population $P(t) = \{x_1(t), x_2(t),..., x_n(t)\}$, where $x_i(t)$ is a feasible solution, t is an iteration number, and n is size of the population. With each generation , stochastically best solutions will reproduce and replace the false ones.

```
procedure genetic algorithm
    begin
    t=0
    initialize P(t)
    evaluate P(t)
    while ( not termination-condition ) do
    begin
        t=t+1
        select P(t) from P(t-1)
        recombine P(t)
        evaluate P(t)
    end
    end
```

Figure 1 . A Simple Genetic Algorithm

The approaches to implementing a simulation are not few. Our choice has been made for the generational approach, where, at each iteration, stochastically higher evaluated solutions are first selected with replacement to form a new population, and then reproduction operators (crossover and mutation) are used to alter the members of the population.

The crossover combines the features of two parent structures to form two similar offsprings. Crossover operates by swapping corresponding segments of a parent string. For example, if parents are represented by five - dimensional vectors, say $x_1 = (a_1, b_1, c_1, d_1, e_1)$ and $x_1 = (a_2, b_2, c_2, d_2, e_2)$, then crossing the vectors between the second and the fifth components would produce an offspring $(a_1, b_1, c_2, d_2, e_2)$ and $(a_2, b_2, c_1, d_1, e_1)$.

A mutation operator arbitrarily alters one or more components of a selected structure - thus increasing the variability of the population. Any position of each solution vector in the new population undergoes a random change with a probability equal to the mutation rate, which keeps constant throughout the computation process.

A genetic algorithm solving a problem must have five components:

1. A genetic representation of solutions to the problem;
2. A way of creating an initial population of solutions;
3. An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness";
4. Genetic operators altering the composition of offsprings during reproduction ;
5. Values assigned to the parameters used by genetic algorithm (population size, probabilities of applying genetic operators, stopping criteria, etc.).

Suppose a maximization of a function of k variables, $f(x_1,..., x_k) : R^k \rightarrow R^n$ takes place. Suppose that each variable x_i gets values from a domain $D_i = [a_i , b_i] \subset R$. We wish to optimize f by some precision : suppose six decimal places for the variables' values are intended.

Each domain D_i should be divided into $(b_i - a_i)*10^6$ equal size ranges. Let m(i) be the smallest integer such that $(b_i-a_i)*10^6 <= 2^{m(i)} - 1$. Then, a representation having each variable coded as a binary string of length m(i) satisfies the precision requirement. Such a string is interpreted by the following formula :

$x_i = a_i + \text{decimal}(\text{string}_2)* (b_i - a_i)/(2^{m(i)} - 1)$

where $\text{decimal}(\text{string})_2$ represents the decimal value of that binary string.

Each potential solution (chromosome) is represented as a binary string of length

$$m = \sum_i m(i)$$

where the group of m(i) bits maps into a value from range $[a_i, b_i]$.

For initializing the population, a number, pop_size, of chromosomes is randomly set in a bitwise fashion. Another method would be to provide some initial potential solutions.

The algorithm operates as follows : for each generation, it evaluates each chromosome (using the function f on the decoded sequences of variables), selects a new population according to the probability distribution based on fitness values, and recombines the chromosomes in the new population by mutation and crossover operators. After a number of generations, when no further improvement can be noticed , the best chromosome will represent an optimal solution. Practically,after a fixed number of iterations, determined by speed and resource criteria, the algorithm stops.

Let us assume the intention of maximizing the following function (the example is taken over from [11]):

$$f(x_1, x_2, x_3) = 3.5(x_1 - 2.1x_3)^3 - \sqrt{x_1x_2 + \log_2(x_3 + 1)} * \sin^2(x_3 + \Pi)$$

where $-3.0 <= x_1 <= 12.1$, $4.1 <= x_2 <= 5.8$, and $0 <= x_3 <= 50.0$. The required precision is of four decimal places for each variable.

The x_1 domain has length 15.1; the precision required implies that [-3.0,12.1] shall be divided into 15.1*10000 equal size ranges. That means that 18 bits are needed for this part of the chromosome :

$$2^{17} < 151\,000 < 2^{18}$$.

Similarly, for the x_2 part of the chromosome, 15 bits are required, whereas for the x_3 part of the chromosome, 19 bits are required . So, the total length of a chromosome is of 18+15+19=52 bits.

Let us consider a sample chromosome :

0100010010110100001111100101000101010100001000100101

The first 18 bits: 010001001011010000 represent $x_1 = -3.0 + \text{decimal}(010001001011010000)_2 * (12-(-3.0))/(2^{18} - 1) = 1.052425$.

Similarly, the next 15 bits represent $x_2 = 5.755330$, and the last 19 bits represent $x_3 = 32.864857$.So, the chromosome

0100010010110100001111100101000101010100001000100101 corresponds to $(x_1, x_2, x_3) = (1.052426,5.755330,32.864857)$. The fitness value for this chromosome is $f(x_1, x_2, x_3)=-4704.82$.

The function f optimization by a GA needs a population of such chromosomes be generated. All 52 bits of each chromosome are randomly initialized . Each chromosome is evaluated and a new population is formed by selecting the more fitted individuals according to their fitness. Some chromosomes from the new population would undergo reproduction by means of crossover and mutation to form new chromosomes (new solutions).

3. Applications of Genetic Algorithms

This section presents some applications of GAs in process identification and control. But, first, let us see some problems raised by the implementations of GAs.

3.1. General Applications and Practical Problems

Sometimes such problems set back, if not block, the finding of the optimal solutions with the desired precision. Ways of dealing with these problems are also shown.

The original concepts of GAs have been shown to provide near-optimal heuristics for information collection in complex search spaces. They frequently outperform other more direct methods such as gradient descent ones when difficult problems should be coped with , i.e. those methods involving highly non-linear, high dimensional, discrete, multimodal or noisy functions. Gradient descent methods are more efficient in finding out solutions when searching convex function spaces with tight constraints, e.g. continuous, low-dimensional,

unimodal spaces. Empirical simulations have frequently demonstrated the efficiency and robustness of GAs in different optimization tasks (see [12], [13], [6]).

No guarantee, however, exists that GAs will find the global functional optima because (1) the limits of precision of the encoding process can largely affect the solution's accuracy, and (2) the search process does not ergodically cover and search the state space. One main deficiency consists in the precision being limited by the length of the population bit-strings. Methods such as dynamic parameter encoding, have addressed the former issue and have achieved good results on a variety of function optimization problems (see [14]). In the GA domain, the second deficiency has hardly been solved.

GAs are not designed to ergodically sample and cover the state space in a maximally efficient way [15]. Doubt is expressed as to whether they are ergodic after all, but the prime benefit of GAs becomes palpable during each generation when all individuals can be evaluated in parallel, making GAs excellent candidates to a running on fine grained parallel processing hardware.

One of the problems encountered by GA applications is premature convergence of the entire population to a non-global optimum. If a too fast convergence occurs , then the valuable information possessed by a part of the population is often lost. This problem is primarily related to the existence of local optima, and depends on both function characteristics and sampling of the solution space.

In general, most approaches to improving the convergence of GAs brought about some modifications on the selection routine. In [16] and [17] , an approach is proposed , which relaxes this problem by decreasing the speed of convergence during the early stages of population existence.

GAs evince inherent difficulties in performing local search for the numerical applications. As observed by Grefenstette: "Once the high performance regions of the search space are identified by a GA, it may be useful to invoke a local search routine to optimize the members of the final population".

For improving the fine local tuning capabilities of a GA, which is a must for high-precision problems, in [11] it is proposed a special mutation operator whose performance is quite different from the traditional one. This non-uniform mutation operates as follows : as the population ages, bits located further to the right of each sequence coding one variable, have higher probability of being mutated, while those on the left have smaller probability.

For some years, GAs have been experimented for solving real-world problems. In 1980, S.F. Smith created a poker-playing set of rules using genetic breeding. L.B.Brooker created a system where a simulated animal learned how to find its way around a machine "world". J.Koza and M.Keane have adapted genetic techniques to keep a broom balanced. J.Koza has also written on genetic approach to economic modelling. He has been especially interested in having GAs discover scientific equations. T.Bogg and B.Huberman have been using GAs to stabilize somehow the chaos in networks. They have found that if there is chaos caused by having many agents on a network competing with each other for a resource, then rewarding the best predictive agents will stabilize that network.

GAs have been quite successfully applied to optimization problems like wire routing, scheduling, game playing, cognitive modelling, the transportation problem, the travelling salesman problem, adaptive and optimal control, fuzzy control problems, pattern recognition, neural networks, machine learning, etc.

Other applications of GAs include :

- GAs for drawing directed graphs (see [18])
- GAs based approach to partial match retrieval based on hash functions (see [19])
- GAs for automated parameter tuning for interpretation of synthetic images (see [20])
- GAs for solving the linear and non-linear transportation problems (see [21], [22], [23])
- GAs for training feedforward neural networks (see [24], [25], [26]).

3.2. Applications in Process Identification and Control

3.2.1. GAs for Identification of Dynamical Systems

We assume that the process of which dynamics is to be identified can mathematically be described as :

$$A(q^{-1})y(t)=q^{-k}B(q^{-1})u(t)+C(q^{-1})e(t),$$

where y(t) is the output and u(t) the input at time t; e(t) is white noise and q^{-1} is the backward shift operator, i.e. $q^{-1}y(t)=y(t-1)$. The involved polynomials are given by :

$$A(q^{-1})=1+a_1q^{-1}+a_2q^{-2}+...+a_{na}q^{-na}$$

$$B(q^{-1})=b_0+b_1q^{-1}+...+b_{nb}q^{-nb}$$

$$C(q^{-1})=1+c_1q^{-1}+...+c_{nc}q^{-nc}$$

Different identification models will all fit a model of the data :

$$A'(q^{-1})y(t)=q^{-k}B'(q^{-1})u(t)+H'(q^{-1})e'(t),$$

where e'(t) is called the residual. Different methods correspond to different forms of the filter $H'(q^{-1})$. $A'(q^{-1})$ and $B'(q^{-1})$ are defined analogously to $A(q^{-1})$ and $B(q^{-1})$. The unknown polynomial coefficients of the model are collected in a parameter vector p.

For the least squares (LS) method we have $H'(q^{-1}) = 1$, and the unknown polynomial coefficients (a_i', b_j') are determined by minimizing

$$V_{LS}=\sum_{t=m}^{N} e'^2(t)$$

where m = max(na, nb+k), N=number of samples.

For the maximum likelihood (ML) method we have $H'(q^{-1})=C'(q^{-1})$. The parameter vector p is determined as the minimum point of

$$V_{ML}=\sum_{t=1}^{N} e'^2(t)$$

An analytical minimization of V_{ML} (p) is not possible since the dependence on the c_j' parameters is strongly nonlinear. In most cases, a Newton-Raphson method is used for the numerical minimization, but this algorithm could be complex and cumbersome.

Taking into account the robustness of GAs, and the fact that they require little knowledge of the problem itself, these algorithms could be used for the identification of linear systems. This approach consists in finding the best structure, orders and parameters using input-output data of the system.

The main problems that such an approach has to consider are:

1. the codification method
2. the possibility of using an initial population of solutions
3. the definition of the genetic operators. Physical considerations and human experience impose some restrictions which have to be considered. The use of special genetic operators would be compulsory (see [23]).

3.2.2. The Use of GAs for Evolving Neurocontrollers

There are processes of which complex dynamics makes them have time delay, time varying parameters, unknown and variable structures, etc., in such a way that it is almost impossible or very difficult to obtain an analytical model of them.

A neural network (NN) based approach is a valid tool in tackling the process control without an explicit analytical model. The property of NNs for building an adaptive model of a complex plant let this technique be used in control problem-solving where traditional techniques will otherwise fail.

The neurocontroller is included in a closed loop and responds straightforwardly to the process observed behaviour by supplying control parameters. The neurocontroller task will be learning to supply the appropriate control parameters for the desired targets given as input.

One can distinguish three possible ways of using GAs for evolving NNs :

1. the evolution of connection weights
2. the evolution of neural network architecture
3. the evolution of learning rules

Most of the researches have been carried out in the first two directions. The obtained results have shown that using a GA as a replacement of back-propagation does not seem to be competitive with the best gradient method, e.g. quickprop. But GAs seem to be an useful learning method when we deal with discontinuous optimality criteria or discontinuous node transfer functions.

Some attempts have been made to adaptively adjust standard training algorithms' parameters using an evolutionary approach. For example, such parameters may be the learning rate and the momentum of a standard back-propagation algorithm.

A more difficult approach is to evolve learning rules. Paper [27] aims at pursuing this direction of research in two ways. The first one is to optimize a standard unsupervised learning algorithm, such as Kohonen's self-organizing map, and the second one is to evolve general unsupervised learning algorithms.

This work would permit the understanding of the complex relation between evolution and learning, and would also be helpful .

Domains of further research are :

1. the codification method
2. a method for supplying initial potential candidates
3. defining problem specific genetic operators

An example of using GAs for evolving NNs can be found in [28]. This paper proposes the evolution of adaptive noise-tolerant dynamical neural networks, which are recurrent and operate in real time. These networks are used for controlling autonomous robots.

3.2.3. Use of GAs in Analysing Robust Stability Problems

Stability of polynomials is a key issue in the analysis and design of automatic control systems. One basic approach to the robustness of linear systems, considers the characteristic polynomial in the presence of parameter uncertainties.

Consider a system characterized by state space form :

$x'(t)=A(q)x(t)$

where q is a vector of uncertain parameters.

The characteristic polynomial is :

$$P(s,q) = \det[sI - A(q)] = \sum_i a_i(q) * s^i$$

The system is said to be robustly stable if the roots of P(s,q) are contained in Re $s < 0$, for any q. We could apply a GA based optimization scheme to finding the maximum root of P(s,q). If this maximum root is positive then the system is unstable, otherwise it is stable (see [29]).

3.2.4. GAs and Optimal Control

The task of designing and implementing algorithms for optimal control problem-solving is a difficult one. Some researchers have studied the application of GAs to discrete-time optimal control problems. They have proved that the GA applies to more general problems, and appears to be more competitive with search-based methods (see [30]).

3.2.5. GAs and Fuzzy Systems

Fuzzy logic is an innovative technology with a broad range of applications. Fuzzy logic is used for the control of home appliances, video equipment, automobiles, process control and industrial automation.
Some attempts have been made at using GAs in fuzzy systems. We will only report on two applications.

3.2.5.1. The Fuzzy Classifier System

In [31] it is proposed a fuzzy classifier system (FCS) which is motivated by the fuzzy controllers concept, but owes to other attempts the manner in which it creates new rules and adjusts the contribution of the existing rules to the system outputs. The FCS adapts the credit assignment mechanisms of common classifier systems to its use of fuzzy rules. Its fuzzy rules are represented as binary strings on which a GA

operates, and so, they allow for the evolution of adapted sets of rules. So, a GA selects the classifiers for reproduction according to their strength: stronger classifiers are more frequently selected than weaker ones. For more details the interested reader is referred to [32] and [33].

3.2.5.2. GAs for Tuning Fuzzy Logic Controllers

Low cost, small size and reliability are main advantages of fuzzy logic control systems. They also provide the simplest solution and the easiest operation. A fuzzy logic controller is very robust and more performant than a conventional PID controller. It can be implemented on very little silicon surface. Special fuzzy logic chips are available, and they can easily be interfaced to sensors and actuators.

Fuzzy controllers are typically highly non-linear systems. Therefore, a fuzzy controller is difficult to understand and analyse. The key problem with fuzzy logic controllers is that they are difficult to calibrate.

Some approaches of this problem using GAs [34], [35] make the assumption that the mathematical model of the process is available, or that it could be obtained by using some classic identification algorithm. In the proposed fuzzy representation, the knowledge is distributed at three different levels: symbolic rules, numerical weights, and fuzzy linguistic definitions. In order to obtain an optimal fuzzy controller with respect to some performance ratio, the search must be guided at all three levels simultaneously. The GA is used for both extracting the necessary knowledge and tuning the existing knowledge of a fuzzy logic controller.

The genetic operators involved are random mutation, dynamic mutation designed for better local exploration, single crossover on whole genes, and arithmetical crossover averaging selected genes. For evaluation purposes, different objective functions, reflecting various criteria, are used. The results indicate that the modifications made by the GA could lead to optimizing the fuzzy controller both in terms of simplifying the symbolic structure (and thus increasing its comprehensibility) and of optimizing the structure using the numerical weights.

What to do when the process model is not available? In order to cope with this situation an architecture of a hybrid geno-fuzzy control system was introduced in [36]. It is presented in Figure 2. The function of process identification is performed on-line by a GA, called GAI (GA Identifier), which is providing each time step Θ, an estimate of the process model, by minimizing a performance ratio V. This model is used by another GA, called GAT (GA Tuner) which adapts the fuzzy logic controller (FLC), with respect to a performance ratio J. The adaptation takes place simultaneously at all the three levels of knowledge described above. The GAT provides Π, the set of tuned controller parameters (numerical weights, fuzzy definitions, fuzzy rules).

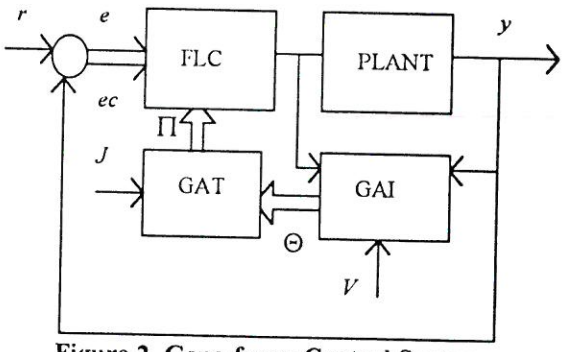

Figure 2. Geno-fuzzy Control System

The functions of identification and tuning could be performed by the same GA, thus resulting a direct geno-fuzzy control system.

3.2.5.3. GAs for Learning the Algebraic Model of the Fuzzy Controller

The large number of parameters to be adaptively modified is a drawback of the previous approach. So, this problem has led the authors of [36] to the idea of aggregating these parameters into a smaller number of parameters. This would result in an easier tuning procedure and would reduce the time necessary for computing the inputs to the process.

The idea was to use for this purpose the algebraic model of the fuzzy controller, introduced in [37]. Suppose that the defuzzified output of the fuzzy controller is δ, where δ is in [-1,1]. If the inputs to the fuzzy controller are r_1, r_2 , ... , the functional relationship between the defuzzified output δ and the inputs, $\delta = F(r_1, r_2, \ldots)$ is called the algebraic model of the fuzzy controller. For example, if the inputs of the fuzzy controller are, as in most of the now cases, the error e and the change of error, ec, the output of the controller would be $\delta = F(e, ec)$. If F is known , and this function is not very complicated, it may be used for computing the process input, and fires no fuzzy rules. Also, this algebraic model would aggregate all the fuzzy controller parameters into a much smaller number of numeric coefficients, thus making the adaptation process faster. In [38] there are reported some situations where the construction of F was possible. This was possible when a small number of rules, and special logic to evaluate the fuzzy control rules, were used. There are some important cases in what regards the structure of F. If $\delta = \alpha * e + \beta * ec$ and α and β are constants, then F is linear. If F has the same structure as above, but α and β are functions of e and/or ec, then the fuzzy controller is a non-linear PI controller. There were reported many experiments on the sensitivity of δ with respect to various parameters, such as the defuzzifier, the fuzzy logic, the scaling constants.

An interesting opportunity to determine the structure of the function F is to use a GA for this purpose, and to further use the algebraic model (AM) of the fuzzy controller for tuning purposes.

[36] introduces an architecture using this approach (see Figure 3). One can observe the GAT and GAI blocks which have the same purposes as in Figure 2. There exists GAM (GA Modeller) which receives as input the look-up table Λ of the fuzzy logic controller (FLC), which is known to perform well on the respective class of processes.

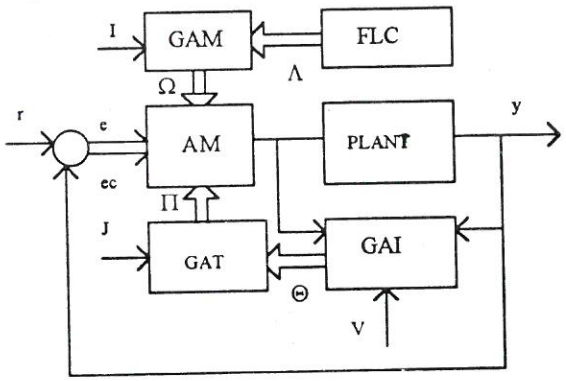

Figure 3 Algebraic Model Based Geno-fuzzy Control System

The GAM performs off-line and provides Ω, an AM of the fuzzy controller, by minimizing some performance ratio I. This AM is further used for controlling the process, and for tuning purposes, based on GAI and GAT.

4. A Case Study

4.1. Test Problem

The experiments consisted in testing the algebraic model based geno-fuzzy control system on a servo-system [36]. The task to be executed was to rotate the shaft of a servomotor to a set point. The fuzzy controller has two inputs, error e and error change ec, and one defuzzified output. The error e is defined as the difference between the set point r and the output y from the shaft encoder : $e = r - y$. The error change is defined as follows: $ec = e_1 - e_2$ where e_1 is the error at the time step t_1, and e_2 is the error at the time step t_2. The input to the process is called u, and it is a voltage output, defined as $u(t) = u(t-\Delta) + \delta(t)$, at times $t = \Delta, 2\Delta, \ldots$, where Δ is the sampling time.

For the linguistic values, there are used the same fuzzy sets for error, error change, and control change specified in [-1,1]. They are presented in Figure 4 . They have the following meaning : SN = "small negative", ZE = "zero", and SP = "small positive".

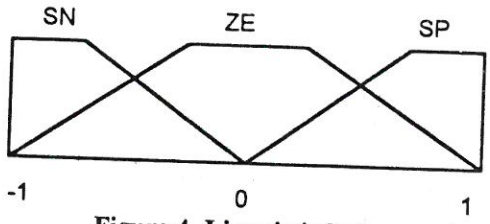

Figure 4. Linguistic Values and Their Fuzzy Sets

Table 1 shows a simple fuzzy controller which contains three fuzzy control rules,relating the output δ to the inputs e and ec. This controller is known to perform well and is further denoted by FC1.

Table 1. Human Expert Generated Rules for FC1

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

First, the corresponding look-up table for FC1 was created, by uniformly quantizing the universe of discourse for e and ec, which is the interval [-1,1], and by computing the corresponding output, using Zadeh logic for inference and the center of gravity method for defuzzifier.

4.2. Genetic Representation

The following model was fitted to data obtained by creating the look-up table of the fuzzy controller.

$$\delta = \frac{g_1 + g_2 * e + g_3 * ec}{g_4 + g_5 * e + g_6 * ec} + \frac{g_7 + g_8 * e + g_9 * ec}{g_{10} + g_{11} * e + g_{12} * ec} + \frac{g_{13} * e * ec}{g_{14} + g_{15} * e + g_{16} * ec} \quad (1)$$

The evaluation function I, used for rating potential solutions, was chosen to be a quadratic index :

$$I = (1/P) \sum_{p=1}^{P} (rp - mp)^2 \quad (2)$$

where p is the *p*th entry of the look-up table, p=1..P, m_p is the output of the potential algebraic model (chromosome) for *e* and *ec* corresponding to this entry and r_p is the output from the *p*th entry of the look-up table. The best algebraic model should have the smallest I. So, in this case the GA operates for minimizing I.

4.3. Genetic Operators

All the operators used in the reported experiments are based on floating point representations of the chromosomes [11].

Uniform mutation is an unary operator and selects a random component (gene) of a selected potential solution (chromosome), which is also selected randomly with a probability equal to the uniform mutation rate. The selected gene is modified at random. Dynamic mutation is an unary operator which is responsible for the fine tuning capabilities of the system. For a parent x, if the element x_k is selected for dynamic mutation, the result is $x' = (x_1,...,x'_k,...,x_n)$, where $x'_k = x_k + D(t,u_k - x_k)$, if a random digit is 0, and $x'_k = x_k + D(t,x_k - l_k)$, if a random digit is 1, and l_k and u_k are the lower and upper limits of x_k. The function D(t,y) returns a value in the range [0,y] such that the probability of D(t,y) being close to 0 increases as t increases (t is the generation number). This property causes this operator to search the space very locally when t is large. An example of D is :

$$D(t,y) = yr(1 - t/T)b \quad (3)$$

where r is a random number in [0,1], T is the maximal generation number, and b is a system parameter (usually b=6).

Simple crossover is the classic crossover operator which is operating by swapping corresponding segments of two parents and producing two offsprings. The whole arithmetical crossover is defined as a linear convex combination of two

parents, i.e. if x_1 and x_2 are chosen to be the parents, the resulting offsprings are $x'_1=ax_1+(1-a)x_2$, and $x'_2=ax_1+(1-a)x_2$, where a is a random number in [0,1].

4.4. Experimental Results

The step response of the servo-system with FC1 is presented in Figure 5. For FC1, the obtained algebraic model is denoted by AM1, and is as follows:

$$\delta = \frac{0.0326 + 3.7279 * e - 0.0025 * ec}{4.9249 + 1.6736 * e + 0.0054 * ec} + \frac{-0.0015 + 0.9076 * e + 0.0099 * ec}{4.4573 + 0.201 * e + 2.8231 * ec} + \frac{1.0359 * e * ec}{4.7052 + 0.0052 * e + 4.0244 * ec} \quad (4)$$

and the corresponding value of the performance ratio was I=0.019756. The step response of the servo-system controlled with AM1 is presented in Figure 6. This model was further tuned by a GA which minimized the following performance ratio.

$$E = \sum e_t^2 \quad (5)$$

where e_t is the error at time t, and is as follows:

$$\delta = \frac{0.04696 + 9.9999 * e + 2.399 * ec}{3.3101 - 9.9999 * e + 0.0054 * ec} + \frac{-0.0015 + 1.0738 * e + 9.9999 * ec}{1.4139 - 0.8834 * e + 0.5255 * ec} + \frac{5.4436 * e * ec}{0.2083 + 3.1907 * e + 8.3747 * ec} \quad (6)$$

The performance ratio for AM1 is E=12.200. The tuned AM1 ensures E=2.404019, and the step response of the control systems with this tuned algebraic model is presented in Figure 7.

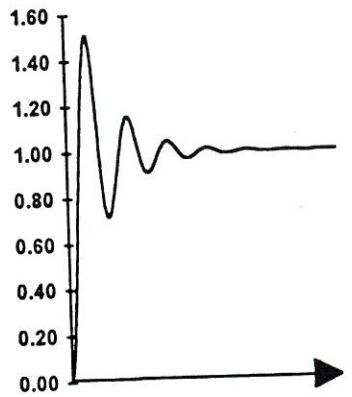

Figure 5. Step Response with FC1

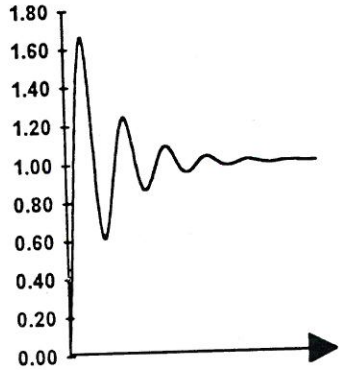

Figure 6. Step Response with AM1

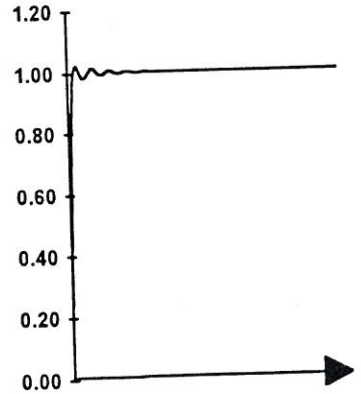

Figure 7. Step Response with Tuned AM1

For both experiments, the population size was 40, the uniform mutation rate was chosen to be 0.1, the dynamic mutation rate was set to 0.15, the simple crossover probability was 0.6, and the arithmetic crossover rate was 0.2. The initial population of solutions was created randomly.

5. Conclusions and Further Research

In this paper the use of genetic algorithms for process identification and control, was discussed. First, we described the theoretical foundations of GAs. Secondly, we concentrated on describing some applications to process identification and control, and on presenting a case study.

GAs require little knowledge of the problem itself. Therefore, computations based on these algorithms are attractive to users with no numerical optimization background. An interesting area of further research is the comparison with related techniques, such as simulated annealing (see [15]).

While the GA methods are likely to be slower in execution than the traditional methods when operating on well-behaved objective functions, moving on to difficult problems compensates much better approximations for the resources waste. This could be reasonable and acceptable if real-time performance were not crucial, as in many cases of practical optimization problems (see [23]). We think that the domain of GAs would strongly benefit from trying to implement these ideas in more complex problems. This would require that, on implementing new genetic operators, knowledge about the problem is available. Another area of interest would be, in our opinion, the design of some methods for providing an initial population of solutions. These methods would be problem-dependent, and will cause a higher operation speed, which could be very useful. Also, implementing new genetic operators for coping with linear and non-linear constraints, is a new research direction . The implementations of GAs on parallel processing computers look very promising.

REFERENCES

1. VON-NEUMANN, J., **Theory of Self-reproducing Automata**, Burks(Ed.), University of Illinois Press, 1966.
2. HOLLAND, J., **A Universal Computer Capable of Executing an Arbitrary Number of Sub-programs Simultaneously**, Proceedings of the EJCC, 1959, pp.108-113.
3. HOLLAND, J., **Adaptation in Natural and Artificial Systems**, University of Michigan Press, Ann Arbor, 1975.
4. DEJONG, K.A., **Genetic Algorithms : A 10-year Perspective**, Proceedings of the First International Conference on Genetic Algorithms, Pittsburgh, PA, 24-26 July, 1985.
5. DAVIS, L., **Genetic Algorithms and Simulated Annealing**, PITMAN, London, 1989.
6. GOLDBERG, D.E., **Genetic Algorithms in Search, Optimization and Machine Learning**, ADDISON-WESLEY, Reading, MA, 1989.
7. BRINDLE, A., **Genetic Algorithms for Function Optimization**, Ph.D dissertation, University of Alberta, Edmonton, Canada, 1981.
8. VALDES, R., **What is Biocomputing?**, DR. DOBB'S JOURNAL, April 1991, pp.46, 108-109.
9. ACKLEY, D.H., **An Empirical Study of Bit Vector Function Optimization**, in L.Davis (Ed.) Genetic Algorithms and Simulated Annealing, PITMAN, London, pp.170-204.
10. ACKLEY, D.H., **Stochastic Iterated Genetic Hill-climbing**, Ph.D Thesis, Dept. of Computer Science, Carnegie Mellon University, 1987.
11. MICHALEWICZ, Z. and JANIKOW, C.Z., **Genetic Algorithms for Numerical Optimization**, STATISTICS AND COMPUTING, 1, 1991, pp.75-91.
12. BETHKE, A.D., **Genetic Algorithms as Function Optimizers**, Ph.D Thesis, Dept. of Computer and Communication Sciences, University of Michigan, 1981.
13. DEJONG, K.A., **An Analysis of the Behavior of a Class of Genetic Adaptive Systems**, Ph.D Thesis, Dept. of Computer and Communication Sciences, University of Michigan, 1981.
14. SCHRAUDOLPH, N.N. and BELEW, R.K., **Dynamic Parameter Encoding for Genetic Algorithms**, Technical Report LAUR 90-2795, Los Alamos National Lab, New Mexico, 1990.

15. INGBER, L. and ROSEN, B., **Genetic Algorithms and Very Fast Simulated Re-annealing: A Comparison**, MATL. COMPUT.MODEL, 16, 1992, pp.87-100.

16. JANIKOW, C.Z. and MICHALEWICZ, Z., **Specialized Genetic Algorithms for Numerical Optimization Problems**, Proceedings of the International Conference on Tools for Artificial Intelligence, Washington, 6-9 November 1990, pp.798-804.

17. JANIKOW, C.Z. and MICHALEWICZ, Z., **Convergence Problem in Genetic Algorithms**, Submitted for publication.

18. GROVES, L.J., MICHALEWICZ, Z., ELIA, P.V. and JANIKOW, C.Z., **Genetic Algorithms for Drawing Directed Graphs**, Proceedings of the Fifth International Symposium on Methodologies of Intelligent Systems, Knoxville, TN,25-27 October 1990, pp.268-276.

19. VALENZUELA-RENDON, M., GUERRA-SALCEDO, C. and ICAZA, J.I., **A Genetic Algorithm Approach to Partial Match Retrieval Based on Hash Functions**, Proceedings of the 4th International Symposium on Artificial Intelligence,1991, pp.156-162.

20. MONTANA, D.J., **Automated Parameter Tuning for Synthetic Image Interpretation**, in L.Davis (Ed.) The Genetic Algorithms Handbook, pp.282-311.

21. VIGNAUX, G.A. and MICHALEWICZ, Z., **A Genetic Algorithm for the Linear Transportation Problem**, IEEE TRANS. ON SYSTEMS, MAN AND CYBERNETICS, Vol.21, No.2, March/April 1991, pp.445-452.

22. MICHALEWICZ, Z., VIGNAUX, G.A. and HOBBS, M., **A Non-standard Genetic Algorithm for the Nonlinear Transportation Problem**, ORSA JOURNAL ON COMPUTING , Vol.3, 1991.

23. MICHALEWICZ, Z. and JANIKOW, C.Z., **GENOCOP : A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints**, COMMUNICATIONS OF THE ACM, 1992.

24. DAVIS, L., **Mapping Classifier Systems into Neural Networks**, Proceedings of the Conference on Neural Information Processing Systems, MORGAN KAUFMANN, 1988.

25. WHITLEY, D., **Applying Genetic Algorithms to Neural Network Problems**, International Neural Network Society, 1988, p.230.

26. MONTANA, D.J. and DAVIS, L., **Training Feedforward Neural Networks Using Genetic Algorithms**, Proceedings of the IJCAI, 1989, pp.762-767.

27. DUMITRACHE, I. and BUIU, C., **Evolutionary Synthesis of Unsupervised Learning Algorithms**, Proceedings of the SPANN 94 Conference, Lille, France, April 1994.

28. HARVEY, I., HUSBANDS, P. and CLIFF, D., **Issues in Evolutionary Robotics**, CSRP 219, July 1992.

29. MURDOCK, T.M., **Use of A Genetic Algorithm to Analyse Robust Stability Problems.**

30. MICHALEWICZ, Z., KRAWCZYK, J.B., KAZEMI, M. and JANIKOW, C.Z., **Genetic Algorithms and Optimal Control Problems**, Proceedings of the 29th Conference on Decision and Control, Honolulu, Hawaii, December 1990, pp.1664-1666.

31. VALENZUELA-RENDON, M., **The Fuzzy Classifier System : Motivation and First Results**, in H.-P.Schwefel & R.Manner (Eds.) Parallel Problem Solving from Nature, SPRINGER-VERLAG, Berlin, 1991, pp. 330-334.

32. VALENZUELA-RENDON, M., **The Fuzzy Classifier System : a Classifier System for Continuously Varying Variables**, Proceedings of the 4th International Conference on Genetic Algorithms,1991, pp.346-353.

33. VALENZUELA-RENDON, M., **Reinforcement Learning in the Fuzzy Classifier System**, International Workshop on Learning Classifier Systems, 1992.

34. DUMITRACHE, I., CALCEV, G., CONSTANTINESCU, R. and BUIU, C., **A Real Time Control Experiment Using A Neural Network Based Linguistic Approach**, AIRTC 92, IFAC/IFIP/IMACS Conference on Artificial Intelligence in Real-Time Control, Delft, The Netherlands, June 16-18, 1992.
35. DUMITRACHE, I., JANIKOW, C.Z. and BUIU, C., **Tuning Fuzzy Logic Controllers Using Genetic Algorithms**, Proceedings of the 9th International Conference on Control Systems and Computer Science, "Politehnica" University of Bucharest, May 25-27, 1993, pp.450-461.
36. BUIU, C. and DUMITRACHE, I., **Genetic Algorithms in Intelligent Control Systems Design**, Preprints of the 2nd IFAC Symposium, SICICA' 94, Budapest, Hungary, June 8-10, 1994, pp. 188-193.
37. BRAAE, M. and RUTHERFORD, D.A., **Theoretical and Linguistic Aspects of the Fuzzy Logic Controller,** AUTOMATICA, 15, 1979, pp. 553-577.
38. BUCKLEY, J.J., **Nonlinear Fuzzy Controller**, INFORMATION SCIENCES, 60, 1992, pp. 261-274.